SEDEM DIVOV INFORMATIKY

SEDEM DIVOV INFORMATIKY

Juraj Hromkovič

Ružomberok 2012

Publikácia vyšla s podporou
Pedagogickej fakulty Katolíckej univerzity v Ružomberku

Recenzenti
Alica Kelemenová, Ján Gunčaga, Monika Steinová, Dennis Komm

Sadzba a preklad
Michal Winczer

Ilustrácie
Ingrid Zámečníková

Grafický návrh obálky
Martin Papčo

Jazyková úprava
Beáta Murínová

VERBUM – vydavateľstvo Katolíckej univerzity v Ružomberku
Námestie Andreja Hlinku 60, 034 01 Ružomberok
http://ku.sk, verbum@ku.sk, tel. +421 44 430 46 93

ISBN 978-80-8084-958-0

Pre

Ursa Kirchgrabera

Burkharda Moniena

Adama Okrúhlicu

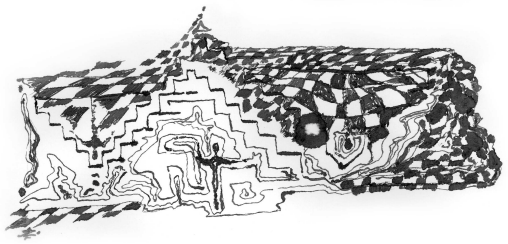
Péťu a Petra Rossmanithových

Georga Schnitgera

Ericha Valkemu

Klausa a Petra Widmayerových

a všetkých, ktorých nadchol výskum



Veda je vo svojej podstate nedeliteľným celkom.
Jej rozdelenie na jednotlivé oblasti
nevyplýva z jej podstaty,
ale je skôr dané hranicami
ľudských schopností v procese poznania.

Max Planck

Predslov

Kniha je výsledným materiálom cyklu prednášok „Sedem divov informatiky“, ktoré sme v zimnom semestri 2005/06 ponúkli širokej verejnosti na ETH v Zürichu. Mnohí ľudia spájajú informatiku len s počítačom a schopnosťou s ním zaobchádzať. Hľadanie informácií na internete, spracovávanie textov a obrázkov patria k základným témam kurzov pre počítačový vodičský preukaz a vo viacerých krajinách sa ich výučba nesprávne považuje za vyučovanie informatiky. Pritom schopnosť používať rôzne softvérové systémy má s informatikou spoločné asi toľko, ako šoférovanie auta so strojárstvom. Ak vodiča motorového vozidla nepokladáme automaticky za strojného inžiniera, nemôžeme ani používateľov počítačov pasovať za informatikov. Pôvodným cieľom tohto prednáškového cyklu bolo nahradiť naivné predstavy o informatike obrazom vedeckej disciplíny, ktorá na jednej strane skúma podobne ako matematika a prírodné vedy zákony fungovania tohto sveta, a tým prispieva k budovaniu všeobecného poznania, a na druhej strane využíva získané poznatky k tvorbe rozličných produktov využitím inžinierskych metód.

Aj keď zmena chápania informatiky, najmä v oblasti vzdelávania, ostala pre nás dôležitá a hodná úsilia, počas prípravy prednáškového cyklu sa naše priority menili a pozornosť sme upriamovali stále viac aj na ďalšie ciele. Vznik a vývin informatiky sme chceli vyrozprávať ako pútavý príbeh. Nie ako príbeh nejakej izolovanej vedy, ale ako vednej disciplíny

nerozlučiteľne spätej s inými vednými odbormi, ktorá čerpá z poznatkov a objavov iných vedných oblastí a zároveň tieto obohacuje vlastnými výsledkami. Sme presvedčení, že poznanie informatiky nám pomáha lepšie poznať štruktúru vied vo všeobecnosti a pochopiť dynamiku výskumu. V tejto súvislosti bolo pre nás dôležité poslucháčom ukázať, že pre úspech výskumu sú podstatné nielen na verejnosti známe objavy a vedecké výsledky, ale že mierou vedeckého pokroku je aj vývoj odborného jazyka a s ním spojená pojmotvorba.

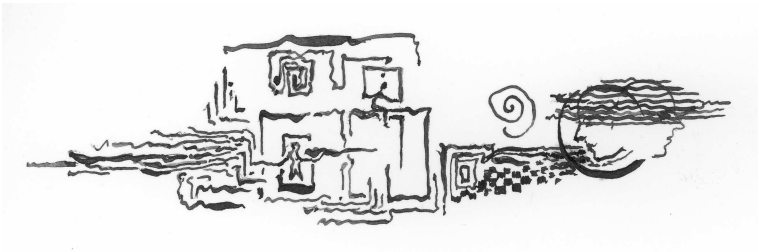
Pri tomto úsilí sme si uvedomili, že nás neuspokojujú typické prednášky pre verejnosť, ktoré sa snažia sprostredkovať význam a dôležitosť vedeckých výsledkov. Zvolili sme si cestu objavov s poslucháčmi, ktorá by ich naučila kráčať samostatne a umožnila im zažiť nadšenie objaviteľov. Nestačí na to len nájsť jednoduchú a názornú reprezentáciu zložitých objektov a vzťahov. Treba pridať ďalší krok a oslobodiť poslucháča z jeho pasívnej úlohy. Preto táto kniha obsahuje aj úlohy pre čitateľa. Úlohy sú v knihe umiestnené tam, kde je aj najzmysluplnejšie ich riešiť. Úsilie ich vyriešiť preverí a upevní správne chápanie predchádzajúcich vysvetlení, alebo vysvetlenia sú výzvou pre čitateľa, aby skúsil vlastnými silami znovuobjaviť pôvodné výsledky výskumu.

Vybrané témy nie sú len míľnikmi vývoja informatiky. Sú aj skutočnými „divmi“ v tom zmysle, že výskumné cesty k nim sú plné neočakávaných zvratov a prekvapujúcich poznatkov, ktoré na prvý pohľad vyzerajú neuveriteľne. Vybrané témy nám umožňujú upútavať poslucháča alebo čitateľa. Vašou úlohou je posúdiť, nakoľko sa to autorovi tejto knihy podarilo.

Želám vám veľa zábavy a vzrušujúci čitateľský zážitok.

Zürich, august 2006. Váš Juraj Hromkovič

Obsah



Žijúc v nepretržitom nadšení
so záujmom o všetko nedosiahnuteľné
rastie človek tým,
že ide neustále vpred.
Zmysel života je v tvorbe
a tvorba je nekonečná.

Maxim Gorkij

Kapitola 1

Stručné dejiny informatiky alebo prečo nie je informatika len vodičským preukazom na riadenie počítača

1.1 Čo sa tu dozvieme?

Cieľ tejto prednášky sa podstatne odlišuje od cieľov nasledujúcich prednášok, ktoré sa vždy venujú jednej špeciálnej téme. V tejto kapitole chceme v populárno-vedeckom jazyku porozprávať príbeh vzniku informatiky ako samostatnej vedeckej disciplíny. Pritom spoznáme nielen základné stavebné kamene informatiky, ale pochopíme do istej miery aj všeobecný proces budovania vedy. Vzhľadom na to, že predstavíme informatiku v kontexte celej vedy, oboznámime sa nielen s niektorými kľúčovými cieľmi základného výskumu v informatike, ale pochopíme aj jej tesnú spätosť najmä s matematikou, ako aj s ostatnými vednými disciplínami. Záverečnú časť kapitoly využijeme na stručné predstavenie obsahu nasledujúcich kapitol v kontexte historického vývoja informatiky.

1.2 Stojí budova vedy na vratkých základoch?

Ak vnímame vedu len ako sumu objavov a vedeckých faktov, dostávame veľmi skreslenú predstavu o vede. Ešte väčšej chyby sa dopustíme, ak vnímame vedné disciplíny len cez ich aplikácie v každodennom živote. Ako by asi vyzerala definícia fyziky, keď je skoro všetko, čo ľudia vytvorili (od domov po stroje až k počítačom), postavené na využití fyzikálnych zákonov? Napriek tomu nepokladáme výrobcov televízorov a počítačov za fyzikov a v nijakom prípade nepasujeme každého, kto vie ovládať televízor a používať počítač, za fyzika. V tomto prípade jasne odlišujeme generovanie poznatkov vo fyzike od ich aplikácií v strojárstve a elektrotechnike. Schopnosť ovládať prístroje s výnimkou počítača nezvykneme spájať so žiadnou vedeckou činnosťou.

Prečo potom mnohí z nás spájajú s informatikou schopnosť pracovať na počítači a používať rôzne softvérové produkty? Akú hodnotu pre všeobecné vzdelanie má schopnosť používať rôzne softvérové systémy (napríklad Word pre spracovanie textov alebo systémy pre spracovanie obrázkov), keď sa tieto dynamicky zásadne menia v priebehu niekoľko málo rokov? Je relatívna zložitosť počítača v porovnaní s inými prístrojmi jediným dôvodom pre toto nedorozumenie?

Isteže, používanie osobných počítačov je tak rozšírené, že počet vodičov motorových vozidiel je približne rovnaký, ako počet užívateľov výpočtovej techniky. No vyučujeme v škole špeciálny predmet vedenie motorových vozidiel? Čoskoro budú z mobilných telefónov malé vysoko výkonné počítače. Stane sa čoskoro používanie mobilov novým predmetom v našich školách? Tieto otázky chcú poukázať na nezmyselnosť zavádzania predmetu informatiky na školách s jediným cieľom naučiť sa používať výpočtovú techniku. Súčasná prax vo viacerých krajinách nám ukazuje, že tieto schopnosti možno nadobudnúť integrovane v ostatných predmetoch a vzhľadom na ich veľmi nízku všeobecnopoznávaciu hodnotu, nemá ich výučba v rámci samostatného predmetu dlhodobú perspektívu.

Ak teda uvažujeme o vyučovaní informatiky, kľúčovou otázkou pre nás je: *„Čo je to informatika?“* Určite pod informatikou nerozumieme schopnosť používať počítač. Ináč by sme sa čoskoro všetci stali informatikmi. Problémy s nejasnými predstavami o tom, čo informatika je a čo nie je, súvisia aj s tým, že informatiku nemôžeme v rámci klasifikácie vied jednoznačne priradiť ani k matematike ako metavede, ani k prírodným vedám a ani k technickým vedám. Situácia s informatikou je podobná, ako keby fyzika, strojárstvo a elektrotechnika boli zlúčené pod jedným náz-

vom v jednom vednom odbore. Vo vývoji softvéru môžeme informatiku považovať za aplikovanú technickú vedu so všetkými znakmi inžinierskych disciplín, ako je vývoj a produkcia zložitých systémov a produktov. Základy informatiky sú ale matematicko-prírodovednej povahy a teoretická informatika hrá pre vývoj softvéru podobnú úlohu ako teoretická fyzika pre technické disciplíny.

Kľúčovou príčinou skreslených predstáv o informatike je práve skutočnosť, že základný výskum[1] v informatike je širokej verejnosti prakticky neznámy. Preto v tejto sérii prednášok venujeme pozornosť hlavne teoretickým základom informatiky.

Uvedomili sme si, teda, že cesta k pochopeniu informatiky nevedie cez štúdium jej aplikácií. Na začiatku sme upozornili aj na to, že veda nie je len súbor vedeckých výsledkov. Preto sú pre nás kľúčové odpovede na nasledujúce otázky:

„Ako vznikajú vedné disciplíny?“

„Čo sú základné kamene vedy?“

Každý vedný odbor má svoj vlastný jazyk, ktorý je postavený na špeciálnych odborných pojmoch (termínoch). Bez nich by nebolo vôbec možné formulovať výroky o objektoch výskumu. Tvorenie pojmov, ako proces určenia významu odborných slov, je kľúčová pre všetky vedy. Definovanie dôležitých základných pojmov tak, aby boli presne a jednoznačne interpretovateľné, stálo vedcov zvyčajne oveľa viac námahy, ako objavenie mnohých všeobecne známych poznatkov. Uveďme niekoľko príkladov. Matematikom trvalo 300 rokov kým sa „dohodli“ na formálne presnej definícii pojmu pravdepodobnosť. Vedci potrebovali vyše 2 000 rokov kým sa im podarilo nájsť zmysluplnú definíciu pojmu nekonečna[2]. Vo fyzike i v hovorovej reči používame často pojem „energia“. Ani vtáčik-letáčik nechyruje čo to je, nieto ešte my. Na celé dejiny fyziky sa môžeme pozerať ako na neukončený príbeh vývoja našich predstáv o tom, čo je to energia. Teraz sa môže niekto prihlásiť a povedať: „Milý pán Hromkovič, čo je veľa, to je veľa. Ja viem, čo je energia. Učil som sa to predsa v škole.“ V tom prípade ale položím nasledujúcu otázku: „Myslíte grécku definíciu energie ako pôsobiacej sily alebo stredoškolskú definíciu energie ako schopnosť fyzikálneho systému vykonávať prácu? Ak chcete tieto definície používať, tak mi povedzte najprv, čo je sila, alebo čo je práca.“

[1] Skúma zákony spracovania informácií a ukazuje hranice toho, čo sa dá automatizovať (vypočítať na počítači).

[2] Túto históriu priblížime v tretej kapitole.

A keď sa o to pokúsite, zistíte, že sa pohybujete v začarovanom kruhu, pretože k definícii pojmov sila a práca používate pojem energia.

Podobne je to i s pojmom života v biológii. Presná definícia tohto pojmu by nám umožnila jednoznačne rozlišovať medzi živou a neživou hmotou. Žiaľ, takúto definíciu na fyzikálno-chemickej úrovni zatiaľ nemáme.

Milí čitatelia! V nijakom prípade vás nechcem zneistiť týmito úvahami. Nie je katastrofou, že nedokážeme presne špecifikovať niektoré používané pojmy. Vo vede pracujeme často s definíciami, ktoré používané pojmy špecifikujú len nepresne, alebo len približne a do istej miery. Patrí to k normálnemu životu výskumníkov. Vedci len musia vedieť, že pri interpretácii výsledkov výskumu nie je možné dosiahnuť vyššiu presnosť, ako je presnosť definícií používaných pojmov. Napredovanie v pojmotvorbe je často hlavnou mierou miery vývinu jednotlivých vedných disciplín. Pekným príkladom pokroku je niekoľkotisícročný vývin nášho chápania pojmu hmoty.

Na pochopenie toho, čo to znamená a ako je ťažké definovať nejaký pojem, uvedieme nasledovný príklad. Uvažujme o slove „stolička“. Stolička nie je abstraktný vedecký objekt. Je to bežný úžitkový predmet a väčšina z nás verí, že vie, čo je stolička. Pokúste sa teda tento pojem zadefinovať pomocou nejakého opisu.

> ***Definovať*** *pojem znamená tak presne opísať objekt, o ktorom uvažujeme, že každý, aj keď ešte nikdy tento objekt (stoličku) nevidel, vie na základe tohto opisu jednoznačne rozhodnúť o každom predmete, či je stoličkou. V definícii smú byť použité len slová, ktorých význam už bol predtým definovaný.*

Pri pokuse definovať pojem stolička môžeme predpokladať, že každý už vie, čo je „noha“ stoličky. V tomto prípade by sme mohli skúsiť začať s tým, že stolička má 4 nohy. Ale počkať, počkať! Má stolička, na ktorej práve sedíte štyri nohy? Nemá náhodou len jednu nohu a k tomu ešte trocha zvláštnu? Radšej to nechajme tak. Mojím cieľom vás nie je potrápiť. Ide len o to, aby sme pochopili, že tvorenie pojmov je nielen kľúčová, ale i veľmi náročná činnosť.

Teraz už vieme, že tvorenie pojmov je kľúčovou úlohou vo vede. Aj samotný vznik informatiky spájame s tvorbou nového pojmu, a to pojmu „algoritmus“. Ale skôr, ako sa začneme zaoberať dejinami informatiky, potrebujeme pochopiť, čo sú to axiómy vo vede.

> ***Axiómy*** *sú základné stavebné kamene vedy. Môžu to byť fakty alebo definície pojmov, o ktorých pravdivosti a korektnosti nemáme žiadne pochybnosti napriek tomu, že sa nedá dokázať ich korektnosť.*

Na prvé počutie to môže znieť nielen zvláštne, ale dokonca i podozrivo. Chceme takto spochybniť spoľahlivosť vedeckých výsledkov a výrokov?

Skúsme túto problematiku najprv objasniť na konkrétnom príklade. Príklad axiómy je, že rozmýšľame správne, takže je nepochybne spoľahlivý aj spôsob našej argumentácie. Môžeme dokázať, že rozmýšľame správne? Ako? Pomocou argumentácie, ktorá stavia na našom spôsobe myslenia? To je nemožné. Neostáva nám teda nič iné, ako dôverovať nášmu spôsobu myslenia. V prípade, že táto axióma nezodpovedá skutočnosti (nie je správna), máme smolu a celá budova vedy sa zrúti, keď z nej odstránime tento základný stavebný kameň. Táto axióma nie je len filozofická. Má aj svoju matematickú podobu, a pretože matematika je formálnym jazykom vedy, nezaobídeme sa bez nej.

Presnejšie teda vysvetlíme, čo znamená naša viera v túto axiómu a ako používame túto axiómu na definovanie korektnej argumentácie nielen v matematike. Kľúčovým bude pre nás pojem implikácie alebo dôsledku. Ak

> *je nejaká skutočnosť B dôsledkom nejakej inej skutočnosti A,*

tak musí platiť:

> *Ak A je pravdivá (A platí),*
> *tak je pravdivá aj skutočnosť B (B platí).*

Inými slovami,

> *nepravda nemôže byť dôsledkom pravdy.*

Pre skutočnosť, že

> B je dôsledkom A

používame v matematike označenie

> $A \Rightarrow B$.

Hovoríme tiež, že „A **implikuje** B“. V tejto terminológii hovorí naša axióma toto:

Ak

> $A \Rightarrow B$ *a* A *platia,*

tak

platí aj B.

Treba upozorniť, že je prípustné, aby nepravda implikovala pravdu. No neprípustné je, aby dôsledkom pravdy bola nepravda.

Na lepšie porozumenie pojmu implikácia a správnej argumentácie uvedieme nasledujúci príklad.

Príklad 1.1 Uvažujme dve tvrdenia A a B.

Tvrdenie A je „*Prší*“

a

tvrdenie B je „*Lúka je mokrá*“.

Predpokladajme, že naša lúka je pod holým nebom (teda nie je zakrytá, alebo zastrešená). Za týchto okolností môžeme akceptovať, že nasledujúce tvrdenie:

„*Keď prší, tak je lúka mokrá*“,

a teda tvrdenie

$A \Rightarrow B$

je pravdivé. Podľa našej interpretácie odborných termínov „dôsledok“ a „implikácia“, musí byť lúka mokrá (musí B platiť), ak prší (ak platí A). Pozrime sa na to ešte detailnejšie.

„*A platí*“ znamená „*Prší*“.
„*A neplatí*“ znamená „*Neprší*“.
„*B platí*“ znamená „*Lúka je mokrá*“.
„*B neplatí*“ znamená „*Lúka je suchá*“.

Vzhľadom na platnosť či neplatnosť tvrdení A a B, existujú štyri možnosti (situácie): S_1, S_2, S_3 a S_4:

S_1: Prší a lúka je mokrá.
S_2: Prší a lúka je suchá.
S_3: Neprší a lúka je mokrá.
S_4: Neprší a lúka je suchá.

Tieto situácie zvykneme opísať pomocou takzvanej pravdivostnej tabuľky (obr. 1.1).

	A	B
S_1	platí	platí
S_2	platí	neplatí
S_3	neplatí	platí
S_4	neplatí	neplatí

obr. 1.1: Pravdivostná tabuľka

Matematici obľubujú zapisovať všetko čo najkratšie a žiaľ sú pritom ochotní aj riskovať, že sa zníži zrozumiteľnosť ich textov pre nematematikov. Matematici označujú platnosť alebo pravdivosť symbolom „1“ a nepravdivosť (neplatnosť) symbolom „0“. Pri tomto označení vyzerá naša pravdivostná tabuľka nasledujúco (obr. 1.2).

	A	B
S_1	1	1
S_2	1	0
S_3	0	1
S_4	0	0

obr. 1.2: Pravdivostná tabuľka v skrátenej forme.

Je dôležité všimnúť si, že platnosť tvrdenia $A \Rightarrow B$ vylučuje len možnosť situácie S_2 v druhom riadku (A platí a B neplatí) tabuľky.

Prvý riadok zodpovedá situácii S_1, v ktorej obe tvrdenia A a B platia. To znamená, že prší a lúka je mokrá. Táto situácia zrejme zodpovedá implikácii $A \Rightarrow B$, a tým aj nášmu očakávaniu. Tým môžeme uzavrieť, že situácia S_1 je pri platnosti implikácia $A \Rightarrow B$ možná (obr.1.3).

Druhý riadok tabuľky „A platí“ a „B neplatí“ zodpovedá situácii S_2, keď prší a lúka je suchá. Táto situácia nie je možná, pretože odporuje platnosti tvrdenia $A \Rightarrow B$. Implikáciu „$A \Rightarrow B$“ chápeme tak, že z pravdivosti (platnosti) A, nevyhnutne vyplýva pravdivosť (platnosť) tvrdenia B. Preto v tabuľke na obr.1.3 označíme situáciu S_2 ako vylúčenú (nemožnú).

Tretí riadok opisuje situáciu S_3, keď neprší (A neplatí) a lúka je mokrá (B platí). Táto situácia môže nastať a neodporuje implikácii $A \Rightarrow B$. Lúka môže byť mokrá aj keď neprší. Mohlo pršať predtým, niekto mohol lúku popolievať, alebo po studenej noci s jasnou oblohou je lúka mokrá z rannej rosy. Z formálneho logického hľadiska implikácia $A \Rightarrow B$ hovorí, čo musí platiť, keď platí A. Ak A neplatí, nekladie implikácia $A \Rightarrow B$

nijaké požiadavky, a preto je v prípade neplatnosti tvrdenia A všetko možné.

Posledný riadok (obe A a B neplatia) zodpovedá situácii, keď neprší (A neplatí) a lúka je suchá (B neplatí). Táto situácia môže nastať a podobne ako S_3 nie je v konflikte s platnosťou implikácie $A \Rightarrow B$.

Na záver si zopakujeme poučenie z tohto príkladu. Keď platí implikácia $A \Rightarrow B$ a platí tiež tvrdenie A („prší“), tak musí platiť aj B („lúka je mokrá“). Ak A neplatí („neprší“), nevyjadruje implikácia „$A \Rightarrow B$“ nijaké požiadavky na B, a teda sú možné obe situácie S_3 (B platí) a S_4 (B neplatí) (obr. 1.3). □

Jediná vylúčená (nemožná) situácia pri platnosti implikácie $A \Rightarrow B$ je situácia S_2:

„A platí a B neplatí“.

Ak nám teda niekto dá pravdivostnú tabuľku pre dve tvrdenia A a B a označí, ktoré situácie sú možné, a ktoré sú nemožné a jediná vylúčená situácia je „A platí a B neplatí“, tak vieme, že platí implikácia $A \Rightarrow B$. Z matematického pohľadu je pravdivostná tabuľka (obr. 1.3) definíciou implikácie. Vo všeobecnosti existuje jednoduché pravidlo na overenie platnosti implikácie $A \Rightarrow B$.

A	B	$A \Rightarrow B$
platí	platí	možná (platí)
platí	neplatí	vylúčená (neplatí)
neplatí	platí	možná (platí)
neplatí	neplatí	možná (platí)

obr. 1.3: Definícia implikácie

Keď vo všetkých možných situáciách, v ktorých A platí, platí aj B, tak platí aj $A \Rightarrow B$

Úloha 1.1 Uvažujme nasledujúce výpovede A a B. A znamená „Je zima“ a B znamená „Hnedé medvede spia“. Implikácia $A \Rightarrow B$ znamená:

„Keď je zima, tak hnedé medvede spia.“

Predpokladajme, že platí implikácia $A \Rightarrow B$. Vytvorte pravdivostnú tabuľku vzhľadom na platnosť A a B a vysvetlite, ktoré situácie sú možné, a ktoré sú vylúčené.

Teraz sme pochopili význam pojmu implikácie a môžeme sa pýtať:

> *Čo má implikácia spoločné s korektnou argumentáciou? Prečo je tento pojem kľúčom ku korektnej argumentácii (dokazovaniu)?*

Pojem implikácia používame na zostavovanie takzvanej **priamej** argumentácie (priamych dôkazov) a **nepriamej** argumentácie (nepriamych dôkazov). Aby sme boli vo ďalšej časti knihy schopní korektne argumentovať, predstavíme obidve tieto základné metódy dokazovania.

Uvažujme znova naše výpovede A („Prší“) a B („Lúka je mokrá“) z príkladu 1.1. Pridajme k nim ešte tretiu výpoveď C („Salamandry sa tešia“). Predpokladajme, že platí implikácia $A \Rightarrow B$ a tiež implikácia:

$B \Rightarrow C$ (*„Keď je lúka mokrá, salamandry sa tešia“*).

Čo je dôsledkom platnosti $A \Rightarrow B$ a $B \Rightarrow C$? Zostavíme si pravdivostnú tabuľku všetkých možných situácií pre platnosť a neplatnosť troch tvrdení A, B a C (obr. 1.4).

	A	B	C	$A \Rightarrow B$	$B \Rightarrow C$
S_1	platí	platí	platí		
S_2	platí	platí	neplatí		vylúčené
S_3	platí	neplatí	platí	vylúčené	
S_4	platí	neplatí	neplatí	vylúčené	
S_5	neplatí	platí	platí		
S_6	neplatí	platí	neplatí		vylúčené
S_7	neplatí	neplatí	platí		
S_8	neplatí	neplatí	neplatí		

obr. 1.4: Pravdivostná tabuľka pre $A \Rightarrow B$, $B \Rightarrow C$

Ak $A \Rightarrow B$ platí, sú situácie S_3 a S_4 vylúčené. Podobne vylučuje platnosť implikácie $B \Rightarrow C$ situácie S_2 a S_6, v ktorých B platí a C neplatí. Pozrime sa na túto tabuľku len z pohľadu tvrdení A a C. Vidíme, že možné sú len situácie:

(i) obe A a C platia (S_1),

(ii) obe A a C neplatia (S_8),

(iii) A neplatí a C platí (S_5, S_7).

Situácie S_2 a S_4, v ktorých A platí a C neplatí, sú vďaka platnosti $A \Rightarrow B$ a $B \Rightarrow C$ vylúčené. Takto dostávame, že implikácia

$A \Rightarrow C$ (*„Keď prší, salamandry sa tešia“*)

platí.

To zodpovedá nášmu očakávaniu. Keď prší, musí byť lúka mokrá ($A \Rightarrow B$). Keď je lúka mokrá, musia sa salamandry tešiť ($B \Rightarrow C$). V konečnom dôsledku spôsobuje dážď, vďaka mokrej lúke, radosť salamandier.

Úvahu

Keď $A \Rightarrow B$ *a* $B \Rightarrow C$ *platia,*
tak platí aj $A \Rightarrow C$

nazývame **priamy dôkaz**. Priame dôkazy môžeme zostaviť z ľubovoľného počtu implikácií. Napríklad platnosť implikácií:

$$A_1 \Rightarrow A_2,\ A_2 \Rightarrow A_3,\ A_3 \Rightarrow A_4,\ \ldots,\ A_{k-1} \Rightarrow A_k$$

nám dovoľuje postulovať platnosť implikácie

$$A_1 \Rightarrow A_k.$$

Priame dôkazy (priame argumentácie) sú teda jednoducho postupnosťami korektných implikácií. Pri výučbe matematiky v školách realizujeme množstvo priamych dôkazov platnosti rôznych matematických viet. Žiaľ, učitelia matematiky nezvyknú upozorňovať žiakov na túto skutočnosť, a preto teraz ukážeme jeden malý príklad z vyučovania matematiky.

Príklad 1.2 Pozrime sa na lineárnu rovnicu $3x-8=4$ o jednej neznámej x. Našou úlohou je dokázať, že

jediné riešenie rovnice $3x - 8 = 4$ je $x = 4$.

Vyjadrenie inými slovami: Chceme ukázať, že platí implikácia:

„Keď $3x - 8 = 4$, potom $x = 4$.“

Označme tvrdenie „$3x-8=4$“ symbolom A a tvrdenie „$x = 4$“ symbolom Z. Aby sme dokázali $A \Rightarrow Z$, potrebujeme postupnosť implikácií, ktorá sa začína s A a končí so Z a o ktorých platnosti nemáme pochybnosti.

Vieme, že rovnosť ostane zachovaná, keď k obom stranám rovnice pripočítame, alebo odpočítame to isté číslo. Pripočítajme k obom stranám rovnice $3x - 8 = 4$ číslo 8. Takto dostaneme:

$$3x - 8 + 8 = 4 + 8$$

a teda

$$3x = 12.$$

Označme tvrdenie $3x = 12$ symbolom B. Práve sme zdôvodnili platnosť implikácie „$A \Rightarrow B$“ („Ak platí $3x - 8 = 4$, potom platí aj $3x = 12$“).

Takto sme získali prvú implikáciu pre našu priamu argumentáciu. Ďalej vieme, že rovnosť ostane zachovaná v prípade, keď vydelíme (alebo vynásobíme) obe strany rovnice tým istým kladným číslom. Vydeľme obe strany rovnice $3x = 12$ číslom 3. Dostávame:

$$\frac{3x}{3} = \frac{12}{3}$$

a teda

$$x = 4.$$

Dokázali sme platnosť implikácie $B \Rightarrow Z$ („Ak platí $3x = 12$, potom platí $x = 4$“).

Platnosť implikácií $A \Rightarrow B$ a $B \Rightarrow Z$ nám umožňuje tvrdiť, že platí implikácia $A \Rightarrow Z$. Takže, „ak platí $3x - 8 = 4$, potom $x = 4$“. Ľahko si overíme, že $x = 4$ spĺňa rovnicu. Môžeme usúdiť, že jediným riešením rovnice $3x - 8 = 4$ je $x = 4$. □

Úloha 1.2 Pomocou priamej argumentácie ukážte, že $x = 1$ je jediné riešenie rovnice $7x - 3 = 2x + 2$.

Úloha 1.3 Pozrime sa na pravdivostnú tabuľku pre tri tvrdenia A, B, C, zobrazenú na obr. 1.5. Vidíme, že možné sú len tri situácie S_1, S_2 a S_8, všetky ostatné

	A	B	C	
S_1	1	1	1	
S_2	1	1	0	
S_3	1	0	1	vylúčené
S_4	1	0	0	vylúčené
S_5	0	1	1	vylúčené
S_6	0	1	0	vylúčené
S_7	0	0	1	vylúčené
S_8	0	0	0	

obr. 1.5: Pravdivostná tabuľka A, B a C z úlohy 1.3

sú z nejakých dôvodov vylúčené. Ktoré implikácie platia? Napríklad $C \Rightarrow A$ platí, pretože, ak v jednej z možných situácií platí C, tak platí aj A. Ktoré ďalšie implikácie ešte platia?

Väčšina ľudí je schopná bez ťažkostí porozumieť priamej argumentácii a úspešne ju uplatňovať. Nepriama argumentácia sa nepovažuje za až tak dobre zrozumiteľnú. Zodpovedanie otázky, či je nepriama argumentácia naozaj o toľko zložitejšia ako priama, alebo či problémy s nepriamou argumentáciou sú aj dôsledkami nedostatočných didaktických prístupov v škole, prenechávame na čitateľa. Vzhľadom na to, že nepriamu argumentáciu budeme potrebovať v kapitolách 3 a 4 k objaveniu niektorých principiálnych poznatkov informatiky, vysvetlíme ju detailne na základnej úrovni, na ktorej možno najspoľahlivejšie dospieť k jej správnemu pochopeniu.

Použijeme opäť náš príklad. Tvrdenie A hovorí („Prší“), B („Lúka je mokrá“) a C („Salamandry sa tešia“). Pre každé tvrdenie D označíme symbolom $\overline{D}$ jeho pravý opak. Teda $\overline{A}$ znamená „Neprší“, $\overline{B}$ znamená „Lúka je suchá“ a $\overline{C}$ znamená „Salamandry sa netešia“. Pravý opak znamená, že keď platí D, tak neplatí $\overline{D}$, a keď platí $\overline{D}$, tak neplatí D. Predpokladajme teraz, že platia implikácie $A \Rightarrow B$ a $B \Rightarrow C$.

Biológovia zistili, že

„salamandry sa netešia“,

teda, že platí $\overline{C}$. Môžeme z toho urobiť nejaký záver?

Keď sa salamandry netešia, nemôže byť lúka mokrá, lebo implikácia $B \Rightarrow C$ zaručuje radosť salamandier, keď je lúka mokrá. Takto vieme s istotou, že platí $\overline{B}$ („lúka je suchá“). Platnosť implikácie $A \Rightarrow B$ a tvrdenia $\overline{B}$ nám teraz zaručuje, že neprší, lebo inak by lúka musela byť mokrá. Takže dostávame, že platí $\overline{A}$. Záverečné pozorovanie je, že z platnosti

$A \Rightarrow B$, $B \Rightarrow C$ a $\overline{C}$

vyplýva platnosť tvrdení

$\overline{B}$ a $\overline{A}$.

Predchádzajúcu úvahu môžeme podporiť formálnou argumentáciou pomocou pravdivostnej tabuľky na obrázku obr. 1.6. Platnosť implikácie $A \Rightarrow B$ vylučuje situácie S_3 a S_4. Platnosť implikácie $B \Rightarrow C$ vylučuje situácie S_2 a S_6. Pretože platí $\overline{C}$ (C neplatí), sú vylúčené aj situácie S_1, S_3, S_5 a S_7. Teda jediná možná situácia je S_8. S_8 znamená, že všetky tri tvrdenia A, B a C neplatia a teda, že platia $\overline{A}$, $\overline{B}$ a $\overline{C}$.

Úloha 1.4 Uvažujme znovu tvrdenia A, B a C. Predpokladajme, že platia $A \Rightarrow B$, $B \Rightarrow C$ a $\overline{B}$. Aké závery z toho možno vyvodiť? Načrtnite pravdivostnú

	A	B	C	$A \Rightarrow B$	$B \Rightarrow C$	C neplatí
S_1	platí	platí	platí			vylúčené
S_2	platí	platí	neplatí		vylúčené	
S_3	platí	neplatí	platí	vylúčené		vylúčené
S_4	platí	neplatí	neplatí	vylúčené		
S_5	neplatí	platí	platí			vylúčené
S_6	neplatí	platí	neplatí		vylúčené	
S_7	neplatí	neplatí	platí			vylúčené
S_8	neplatí	neplatí	neplatí			

obr. 1.6: Pravdivostná tabuľka A, B a C

tabuľku pre všetkých 8 situácií, vzhľadom na platnosť tvrdení A, B a C. Určte, ktoré situácie sú možné, keď platia tvrdenia $A \Rightarrow B$, $B \Rightarrow C$ a $\overline{B}$.

Všimnime si, že z platnosti $A \Rightarrow B$, $B \Rightarrow C$ a C nemožno povedať nič o platnosti A a B. Ak platí C, znamená to, že salamandry sa tešia. Ale to neznamená, že lúka musí byť mokrá. Salamandry sa môžu tešiť aj z iných dôvodov. Mokrá lúka je iba jeden z možných dôvodov ich radosti.

Úloha 1.5 Načrtnite pravdivostnú tabuľku pre A, B a C a zistite, ktoré situácie sú možné, ak platí $A \Rightarrow B$, $B \Rightarrow C$ a C.

Úloha 1.6 Uvažujme o nasledujúcich výpovediach C a D. C znamená „Zmiešali sme žltú a modrú farbu“ a D znamená „Miešaním sme dostali zelenú farbu“. Implikácia $C \Rightarrow D$ znamená:

> „Ak zmiešame žltú a modrú farbu, dostaneme zelenú farbu“.

Predpokladajme, že $C \Rightarrow D$ platí. Načrtnite pravdivostnú tabuľku pre C a D a vysvetlite, ktoré situácie sú možné a ktoré vylúčené.
Je možné usúdiť, že z platnosti $C \Rightarrow D$ platí nasledujúce tvrdenie?

> „Ak miešaním nevznikla zelená farba, tak sme nezmiešali modrú a žltú farbu“.

Pomaly, ale isto začíname chápať ideu nepriamej argumentácie. Pri priamych dôkazoch vieme, že nejaké tvrdenie A platí a chceme dokázať platnosť takzvaného cieľového tvrdenia Z. Na dosiahnutie nášho cieľa Z vytvoríme postupnosť korektných implikácií:

$$A \Rightarrow A_1,\ A_1 \Rightarrow A_2,\ \ldots,\ A_{k-1} \Rightarrow A_k,\ A_k \Rightarrow Z,$$

ktoré spoločne zaručujú platnosť implikácie $A \Rightarrow Z$. Z platnosti A a $A \Rightarrow Z$ môžeme potom usúdiť, že Z platí.

Nepriamy dôkaz má nasledovnú štruktúru:
Východisková situácia: D platí
Cieľ: Z platí

Začneme budovať postupnosť implikácií zo $\overline{Z}$ (teda z opaku toho, čo chceme dokázať) s cieľom dokázať platnosť implikácie $\overline{Z} \Rightarrow \overline{D}$, pričom vieme, že D neplatí. Nech postupnosť implikácií vyzerá nasledovne:

$$\overline{Z} \Rightarrow A_1,\ A_1 \Rightarrow A_2,\ \ldots,\ A_{k-1} \Rightarrow A_k,\ A_k \Rightarrow \overline{D}.$$

Z tejto postupnosti implikácií môžeme usúdiť, že neplatí $\overline{Z}$, a teda platí Z. Správnosť tohto záveru môžeme pozorovať v pravdivostnej tabuľke na obr. 1.7. Situácia S_2 je vylúčená vďaka platnosti $\overline{Z} \Rightarrow \overline{D}$. Pretože platí

	D	Z	$\overline{D}$	$\overline{Z}$	$\overline{Z} \Rightarrow \overline{D}$	D platí
S_1	platí	platí	neplatí	neplatí		
S_2	platí	neplatí	neplatí	platí	vylúčené	
S_3	neplatí	platí	platí	neplatí		vylúčené
S_4	neplatí	neplatí	platí	platí		vylúčené

obr. 1.7: Pravdivostná tabuľka pre D a Z

D, sú vylúčené aj situácie S_3 a S_4. V jedinej možnej ostávajúcej situácii S_1 platí Z, a tým sme dosiahli náš cieľ.

Túto metódu dôkazu nazývame nepriamou, pretože v postupnosti implikácií argumentujeme odzadu smerom dopredu. Ak neplatí $\overline{D}$ (teda ak platí D), tak nemôže platiť ani $\overline{Z}$, a teda platí Z. V našom predchádzajúcom príklade bolo tvrdenie D tvrdením $\overline{C}$, to znamená, že naša východisková situácia bola, že salamandry sa netešia. Chceli sme ukázať, že potom neprší, t.j. náš cieľ bola platnosť $Z = \overline{A}$. V novom označení $Z = \overline{A}$ a $D = \overline{C}$ boli zodpovedajúce implikácie

$$A \Rightarrow B,\ B \Rightarrow C$$

$$\overline{Z} \Rightarrow B,\ B \Rightarrow \overline{D}.$$

Z týchto implikácií sme odvodili platnosť implikácie $\overline{Z} \Rightarrow \overline{D}$. Z platnosti $\overline{Z} \Rightarrow \overline{D}$ a D sme dospeli k záveru, že musí platiť opak tvrdenia $\overline{Z} = A$. Opak tvrdenia $\overline{Z}$ je tvrdenie $Z = \overline{A}$, a tak sme dokázali, že neprší (platí $\overline{A}$).

Vo všeobecnosti postupujeme v nepriamych dôkazoch nasledovne. Chceme

ukázať, že platí nejaké tvrdenie Z. Hľadáme takú postupnosť implikácií

$$\overline{Z} \Rightarrow A_1,\ A_1 \Rightarrow A_2,\ \ldots,\ A_k \Rightarrow U,$$

ktorá sa začína so $\overline{Z}$ a končí sa s nejakým „nezmyslom U“. Za „nezmysel“ považujeme tvrdenie, ktoré očividne neplatí, alebo o ktorom sme už ukázali, že je neplatné. Vďaka platnosti

$$\overline{Z} \Rightarrow U,$$

vidíme, že dôsledkom $\overline{Z}$ je evidentná nepravda (nezmysel). Keďže táto nepravda nemôže platiť, nemôže platiť ani tvrdenie $\overline{Z}$, ktorého je táto nepravda dôsledkom. Musí teda platiť opak tvrdenia $\overline{Z}$, čo je tvrdenie Z.

Úloha 1.7 Nech x^2 je nepárne číslo. Našou úlohou je prostredníctvom nepriamej argumentácie ukázať, že aj x musí byť nepárne číslo. Označme symbolom A tvrdenie, že „x^2 je nepárne“ a symbolom Z tvrdenie, že „x je nepárne“. Implikáciu $\overline{Z} \Rightarrow \overline{A}$, „$x$ je párne“ $\Rightarrow$ „x^2 je párne“) môžeme dokázať umocnením ľubovoľného párneho čísla $2i$ na druhú, čo zapíšeme takto:

$$(2i)^2 = 2^2 i^2 = 4i^2 = 2(2i^2)$$

Výsledné číslo $(2i)^2$ je zrejme deliteľné dvoma bezo zvyšku, a teda je párne.

Doplňte argumentáciu nepriameho dôkazu.

Úloha 1.8 Nech x^2 je párne číslo. Dokážte pomocou nepriamej argumentácie, že x je párne číslo.

Úloha 1.9 (tvrdý oriešok) Dokážte pomocou nepriamej argumentácie, že $\sqrt{2}$ nie je racionálne číslo. Racionálne čísla sú čísla, ktoré sa dajú reprezentovať ako podiel dvoch celých čísel.

V podstate axiómy korektnej argumentácie môžeme považovať za vytvorenie pojmu implikácia vo formálnom spôsobe myslenia. Axiómy často nepredstavujú nič iné ako definície presného významu istých pojmov. V kapitole 3 sa napríklad zoznámime s definíciou pojmu nekonečna, ktorá naše predstavy o nekonečne matematicky spresňuje. Pravda neexistuje možnosť dokázať, že tieto definície dôležitých pojmov zodpovedajú presne našim predstavám. Ale na druhej strane existuje možnosť axiómu vyvrátiť. Jedna možnosť je ukázať, že axióma je v rozpore s inými etablovanými axiómami alebo poznatkami. Druhá možnosť je ukázať, že axióma nezodpovedá našim predstavám, i keď sme si to najprv mysleli. Napríklad, niekto môže nájsť nekonečný objekt, ktorý nie je podľa našej definície nekonečný, a potom musíme túto definíciu revidovať. Revíziu nejakej axiómy

alebo nejakej definície nesmieme pokladať za nešťastie alebo nebodaj dokonca za katastrofu. Výmena alebo úprava jedného zo základných kameňov vedy môže síce viesť k rozsiahlym zmenám, a tým k množstvu práce, ale v konečnom dôsledku je to radostná udalosť, pretože po rekonštrukcii je budova vedy o kus stabilnejšia a bohatšia. Väčšina axiomatických základov matematiky má už za sebou turbulentné časy revízie tvorenia pojmov. Súčasná matematika pôsobí vďaka tomu až príliš stabilne, čoho negatívnym dôsledkom je prehliadanie kľúčovej úlohy tvorenia pojmov najmä vo výučbe matematiky.

Doteraz sme hovorili iba o základných kameňoch vedy. Čo ale môžeme povedať o kameňoch, ktoré nepokladáme za tie základné? Vedci sa usilujú budovať vedu tak, aby korektnosť základných kameňov vo forme axióm automaticky zaručovala korektnosť celej stavby. To je tá všeobecne známa vecnosť a spoľahlivosť exaktných vied. Keď sedia axiómy, tak sedia aj všetky výsledky a z nich odvodené poznatky.

1.3 Vznik informatiky ako koniec jednej eufórie

Koncom devätnásteho storočia ľudia žili v eufórii v dôsledku úspechov vedy a techniky. Technická revolúcia transformovala nové poznatky do vývoja rozličných strojov a zariadení. Produkty kreatívnej práce vedcov a inžinierov si nachádzali cestu do každodenného života a zvyšovali jeho kvalitu. Nepredstaviteľné sa stávalo realitou. Nadšenie z úspechov viedlo vedcov k prílišnému optimizmu, dokonca až k utopistickým predstavám o možnostiach vedy a techniky. V tomto čase prevládala kauzálno-deterministická predstava o svete. Všetko, čo sa odohráva, musí mať príčinu a každá príčina má jednoznačne determinované dôsledky. Reťazcom

$$\begin{aligned}
\text{príčina}_1 \;&\Rightarrow\; \text{dôsledok}_1 \\
&\qquad\; \text{dôsledok}_1 \;=\; \text{príčina}_2 \\
&\qquad\qquad\; \text{príčina}_2 \;\Rightarrow\; \text{dôsledok}_2 \\
&\qquad\qquad\qquad\; \text{dôsledok}_2 \;=\; \text{príčina}_3 \\
&\qquad\qquad\qquad\qquad\; \text{príčina}_3 \;\Rightarrow\; \ldots
\end{aligned}$$

chceli vedci vysvetliť usporiadanie sveta. Verilo sa, že ľudia sú schopní objaviť všetky prírodné zákonitosti a že tieto vedomosti sú postačujúce na porozumenie všetkého. Vo fyzike sa táto eufória prejavila vo viere v takzvaných „démonov“, schopných vypočítať, a tým aj predpovedať budúcnosť. Fyzici už vtedy vedeli, že vesmír je obrovský a že pozos-

táva z veľkého množstva častíc. Preto im bolo jasné, že nijaký človek nie je schopný obsiahnuť naraz pozíciu a stav všetkých častíc vo vesmíre, nehovoriac o všetkých ich možných interakciách. Fyzici teda videli, že ani so znalosťami všetkých prírodných zákonov nebude človek schopný predpovedať budúcnosť. Preto zaviedli pojem démona ako nadčloveka, ktorý je schopný úplne obsiahnuť súčasný stav vesmíru (stav všetkých častíc a interakcií medzi nimi). So znalosťami všetkých prírodných zákonov by takto démon mal byť hypoteticky schopný vypočítať ďalší vývoj a tým predpovedať budúcnosť. Ja osobne nepovažujem tieto predstavy za optimistické, pretože ich akceptovanie by znamenalo, že budúcnosť je už definitívne určená. Kde by potom bolo miesto pre naše aktivity? Nie sme schopní nič ovplyvniť, v najlepšom prípade len predpovedať? Našťastie tieto predstavy rozbili samotní fyzici. Na jednej strane teória chaosu a dynamických systémov ukázala, že existujú reálne systémy, pri ktorých nemerateľne malé rozdiely vo východiskovej situácii môžu viesť k úplne odlišnému vývoju. Definitívnu bodku za predstavami a existencii démonov urobila kvantová fyzika[3], ktorá sa stala základom modernej fyziky. Kľúčovým princípom kvantovej mechaniky je existencia náhodných javov, a tým nepredpovedateľnosť udalostí na úrovni častíc. Keď akceptujeme zákony kvantovej fyziky (doteraz súhlasili všetky experimentálne výsledky s predpoveďami založenými na teoretických výpočtoch), tak neexistuje jednoznačná budúcnosť a nestrácame priestor pre tvarovanie našej budúcnosti.

Založenie informatiky súviselo ale s inými utopistickými predstavami. David Hilbert, jeden z najvplyvnejších matematikov svojich čias, veril v existenciu **metód na riešenie** všetkých problémov. Jeho predstavy o existencii „perfektnej“ matematiky boli nasledovné:

(i) Celú matematiku možno vybudovať na konečnom počte axióm.

(ii) Takto vybudovaná matematika bude úplná v tom zmysle, že o všetkých tvrdeniach formulovateľných v jazyku tejto teórie je možné rozhodnúť na základe argumentácie v tejto teórii, či sú pravdivé alebo nepravdivé.

(iii) Na vytvorenie dôkazov pravdivosti matematických tvrdení existuje metóda, ktorou sa dá táto činnosť matematikov automatizovať.

Náš záujem teraz sústredíme na pojem **metóda**. Čo vo svojej dobe chápali matematici pod týmto pojmom?

[3]Podrobnejšie sa touto témou budeme zaoberať v kapitolách o náhode a kvantových počítačoch.

> *Metóda na riešenie nejakej úlohy je opis postupu riešenia, ktorý vedie k správnemu výsledku. Opis pozostáva z postupnosti inštrukcií, ktoré dokáže realizovať každý (aj nematematik).*

Dôležité je si pritom uvedomiť, že na to, aby nejaká metóda bola aplikovateľná, nepotrebujeme rozumieť, ako bola objavená a prečo úspešne rieši danú úlohu. Ako príklad si zoberme úlohu (problém) riešenia kvadratických rovníc tvaru

$$x^2 + bx + c = 0.$$

Ak $b^2 - 4c > 0$, opisujú formuly (vzorce)

$$x_1 = -(\tfrac{b}{2}) + \tfrac{\sqrt{b^2-4c}}{2}$$
$$x_2 = -(\tfrac{b}{2}) - \tfrac{\sqrt{b^2-4c}}{2}$$

dve riešenia tejto kvadratickej rovnice. Vidíme, že sme schopní vypočítať riešenia x_1 a x_2 bez toho, aby sme vedeli, ako boli odvodené a prečo uvedené vzorce vyzerajú práve takto. Úplne stačí, ak vieme realizovať aritmetické operácie. Vďaka tomu môže aj počítač (dokonca programovateľná kalkulačka), stroj bez akéhokoľvek intelektu, riešiť touto metódou kvadratické rovnice.

Z tohto dôvodu spájame existenciu nejakej metódy na riešenie istého typu úloh s **automatizovateľnosťou** riešenia týchto úloh. Dnes sa pojem metódy na popis postupu riešenia bežne nepoužíva, pretože má v kontexte rôznych vedných oblastí rôzne a veľmi široké interpretácie. Namiesto toho používame pojem **algoritmus**, ktorý je kľúčovým pojmom informatiky[4]. Odborný termín „algoritmus“ vďačí za svoje meno arabskému matematikovi Al-Chwárizmímu, ktorý v deviatom storočí napísal v Bagdade knihu o algebraických metódach.

V zmysle takejto algoritmickej interpretácie môžeme označiť Hilbertove úsilie za snahu automatizovať (algoritmizovať) prácu matematikov. Hilbert hľadal dokonalú matematiku, v ktorej by sme mali algoritmus (metódu) na overovanie pravdivosti sformulovaných tvrdení. Takto by bola automatizovateľná hlavná úloha matematikov, objavovať matematické dôkazy. V podstate ide o neradostnú predstavu, podľa ktorej by bolo možné nahradiť „hlúpym“ strojom jednu z najkreatívnejších intelektuálnych činností.

[4] hoci je tento pojem relatívne nový, algoritmy, v zmysle metód na riešenie (číselných) úloh, používame už tisíce rokov.

Našťastie v roku 1931 dal Kurt Gödel definitívnu bodku za úsilím vytvoriť úplnú matematiku. Gödel matematicky dokázal, že úplná matematika podľa Hilbertových predstáv nemôže existovať, a teda nemá zmysel sa ju snažiť budovať. Najdôležitejšie Gödelove výroky pre vedu môžeme zhrnúť bez použitia matematických formulácií takto:

(a) Neexistuje úplná „zmysluplná“ matematická teória. V každej korektnej a dostatočne obsiahlej matematickej teórii vybudovanej s konečným počtom axióm (ako napríklad v dnešnej matematike) je možné formulovať tvrdenia, ktorých korektnosť nie je možné dokázať v rámci tejto teórie. Na dokázanie alebo vyvrátenie korektnosti týchto tvrdení je nevyhnutné prijať nové axiómy, a tým rozšíriť doterajšiu teóriu.

(b) Neexistuje metóda (algoritmus) na automatické dokazovanie matematických viet.

Ak Gödelove výsledky správne interpretujeme, uvedomujeme si, že jeho objavy sú vlastne pozitívne. Budovanie matematiky ako formálneho jazyka vedy je nekonečný proces. S každou novou axiómou, a tým aj s každým novým pojmom, rastie náš slovník a sila argumentácie. Vďaka novým axiómam a s nimi spojenými pojmami (odbornými termínmi) môžeme formulovať výpovede o objektoch a javoch, o ktorých sme predtým nevedeli hovoriť. Navyše môžeme skúmať a overovať platnosť tvrdení, ktoré boli predtým neoveriteľné. A nakoniec, proces overovania platnosti tvrdení nemôžeme automatizovať. V tom je ľudská tvorivá činnosť nenahraditeľná.

Gödelove výsledky zmenili náš pohľad na vedu. Dnes čoraz väčšmi chápeme vývoj jednotlivých vedných disciplín ako proces tvorenia pojmov a vývoja metód. Prečo ale boli Gödelove výsledky kľúčové pre vznik informatiky? Jednoducho preto, že pred Gödelovymi objavmi nikto nepociťoval potrebu formálne presne (matematicky) definovať pojem metódy. Nikto nepotreboval takúto definíciu na to, aby mohol prezentovať nejakú novú metódu na riešenie istých úloh. Postačilo intuitívne chápanie metódy v zmysle jednoduchého a zrozumiteľného opisu postupu riešenia. Ale vo chvíli, keď vznikla potreba ukázať, že pre isté úlohy neexistujú metódy (algoritmy) na ich riešenie, museli vedci najprv presne špecifikovať, čo pojem metódy (algoritmu) zahrňuje. Je prakticky nemožné dokázať neexistenciu nejakého objektu, ktorý nie je presne špecifikovaný (opísaný). Najprv musíme vedieť úplne presne v zmysle matematickej definície, čo je a čo nie je algoritmus (metóda) na riešenie nejakého problému. Len tak máme šancu pokúšať sa nájsť dôkaz tvrdenia, že pre istú konkrétnu úlohu neexistuje nijaký algoritmus na jej riešenie. Prvú matematickú definí-

ciu algoritmu (pojmu metóda) sformuloval v roku 1936 Alan Turing pomocou takzvaného Turingovho stroja. Neskôr nasledovali mnohé ďalšie definície. Najdôležitejšie je, že všetky rozumné pokusy nájsť formálnu definíciu algoritmu viedli k tomu istému pojmu v zmysle automaticky riešiteľného. Hoci vyjadrené rôznymi matematickými formalizmami, zodpovedajúce definície množín algoritmicky riešiteľných úloh ostali stále tie isté. To viedlo nakoniec k tomu, že Turingova definícia algoritmu bola pridaná k predchádzajúcim už etablovaným axiómam matematiky[5].

Teraz môžeme znova preveriť naše chápanie axióm. Definíciu algoritmu považujeme za axiómu, pretože neexistuje možnosť dokázať jej správnosť. Ako možno dokázať, že naša definícia automatickej (algoritmickej) riešiteľnosti skutočne zodpovedá našim intuitívnym predstavám o automatickej riešiteľnosti? Na druhej strane nemôžeme vylúčiť možnosť, že sa ukáže, že táto axióma je nesprávna. Ako sa to môže stať? Ak niekto vymyslí na riešenie nejakého problému aplikovateľnú metódu, a táto metóda podľa našej definície nie je algoritmom, potom nie je naša definícia dosť dobrá a potrebuje revíziu. Skutočnosťou ale je, že napriek mnohým pokusom sa nikomu nepodarilo vyvrátiť Turingovu definíciu z roku 1936. Dnes pokladáme definíciu algoritmu za stabilnú axiómu a viera v jej správnosť je veľmi silná.

Pojem algoritmu je pre informatiku natoľko kľúčový, že vytvoreniu a pochopeniu pojmov algoritmus a program venujeme radšej celú nasledujúcu kapitolu.

1.4 História informatiky a koncepcia knihy

Prvá základná otázka informatiky bola:

> *Existujú úlohy, ktoré sa nedajú algoritmicky (automaticky) riešiť? A ak existujú, ktoré úlohy sú algoritmicky riešiteľné a ktoré nie sú?*

Naším cieľom je nielen dať odpovede na uvedené otázky, ale aj predviesť veľké úseky výskumných ciest vedúcich k ich zodpovedaniu takým spôsobom, aby ste po nich mohli samostatne kráčať. Vzhľadom na to, že táto téma patrí medzi najťažšie v prvých dvoch rokoch univerzitného štúdia informatiky, budeme postupovať pomaly, krôčik za krôčikom. Z toho dôvodu venujeme tejto najstaršej oblasti informatiky celé tri kapitoly.

[5] Všetky axiómy matematiky sú aj axiómy informatiky.

Druhá kapitola má názov

> *Algoritmika alebo čo má spoločné programovanie a pečenie koláča?*

a venuje sa plne budovaniu kľúčových pojmov algoritmus a program. Aby sme získali prvú predstavu o význame týchto pojmov, začneme so známou činnosťou – pečením koláča.

Piekli ste už podľa receptu koláč, alebo ste niekedy varili jedlo bez toho, aby ste vedeli, prečo sa postupuje práve tak, ako je to napísané v recepte? Celý čas ste si boli vedomí, že korektné vykonávanie jednotlivých pokynov je dôležité pre dosiahnutie dobrej kvality konečného produktu. Čo ste sa pritom naučili? Pomocou presne sformulovaných pokynov detailne napísaného receptu môžeme dospieť k dobrému jedlu aj bez toho, aby sme boli kuchárskymi majstrami. Aj keď sa v eufórii z kuchárskeho úspechu môžeme na okamih pokladať za vynikajúcich kuchárov, nie sme nimi, pokiaľ do hĺbky nerozumieme vplyvu jednotlivých úkonov v recepte na kvalitu konečného produktu, a tým pádom nie sme sami schopní formulovať podobné recepty.

Počítač to má ešte oveľa ťažšie: Je schopný vykonávať len zopár jednoduchých inštrukcií, akými sú v prípade varenia inštrukcie miešania a zohrievania. Na rozdiel od nás počítač nemá nijakú inteligenciu, teda nie je schopný improvizovať. Počítač dôsledne vykonáva pokyny svojich receptov (ktoré sa nazývajú programy) bez toho, aby tušil aké zložité spracovanie informácií práve vykonáva.

Spolu objavíme, že umenie programovať je umením písať recepty tak, aby v nich reprezentované metódy a algoritmy boli zrozumiteľné pre počítač. Pri tejto príležitosti predstavíme aj samotný počítač a ukážeme, ktoré pokyny (inštrukcie) je schopný vykonávať a čo sa v ňom odohráva pri realizácii týchto inštrukcií. Popritom sa naučíme aj čo sú algoritmické úlohy (problémy) a aký rozdiel je medzi významom pojmov program a algoritmus.

Názov tretej kapitoly je:

> *Nie je nekonečno ako nekonečno, alebo prečo je nekonečno v informatike nekonečne dôležité?*

Táto kapitola sa celá venuje nekonečnu. Prečo bolo v procese poznávania nášho konečného sveta definovanie pojmu „nekonečno“ nielen užitočné, ale dokonca nevyhnutné?

Nám známy vesmír je síce obrovský, ale konečný. Všetko, čo vidíme, všetko čoho sa dotýkame a s čím experimentujeme a všetko, čo ovplyvňujeme je konečné. Nikto nikdy nemal nič dočinenia s niečím nekonečným. A napriek tomu bez dnešnej podoby pojmu nekonečno nemôže existovať matematika, fyzika a informatika, a tým ani ostatné vedy. Už v prvých triedach základnej školy sa stretávame s prirodzenými číslami 0,1,2,3,..., ktorých je nekonečne veľa.

Načo je to vôbec dobré, keď aj počet všetkých elementárnych častíc vo vesmíre je síce obrovské, ale stále len konečné číslo? Načo potrebujeme ešte väčšie čísla? Čo znamená nekonečno pre informatiku? A prečo je dôležité nekonečno pri skúmaní hraníc automatickej (algoritmickej) riešiteľnosti?

Usilujúc sa nájsť odpovede na tieto otázky, zoznámime sa nielen s matematickou definíciou nekonečna, ale aj pochopíme užitočnosť konceptu nekonečnosti. Pochopíme, že na prvý pohľad umelo vyzerajúce nekonečno sa stalo úspešným a dokonca nenahraditeľným nástrojom na skúmanie nášho konečného sveta.

Názov štvrtej kapitoly je:

> *Vypočítateľnosť alebo prečo existujú úlohy, ktoré sa nedajú vyriešiť na programovateľnom počítači?*

Táto kapitola je nadstavbou získaných vedomostí o nekonečne, ktoré využíva na ukázanie existencie algoritmicky neriešiteľných úloh.

Ako môžeme dokázať algoritmickú neriešiteľnosť konkrétnych úloh z praxe? Na dosiahnutie tohto cieľa použijeme metódu redukcie, ktorá je jedným zo základných a najúspešnejších prostriedkov matematiky na riešenie konkrétnych úloh. V istom zmysle umožňuje metóda redukcie rozširovať pozitívne výsledky, pretože transformuje známe algoritmy riešenia konkrétnych úloh na nové algoritmy, ktoré riešia nové, často všeobecnejšie úlohy. My ale modifikujeme metódu redukcie tak, aby sa dala použiť na šírenie negatívnych výsledkov, konkrétne na dokazovanie algoritmickej (automatickej) neriešiteľnosti konkrétnych úloh. Týmto spôsobom objavíme konkrétne úlohy z praxe, ktoré sa v nijakom prípade nedajú automaticky riešiť. Naplníme tým náš prvý cieľ tejto série prednášok, zoznámiť sa s hranicou automatickej riešiteľnosti.

Začiatkom šesťdesiatych rokov 20. storočia vedci úspešne vybudovali teóriu, pomocou ktorej klasifikujeme problémy na automaticky riešiteľné a automaticky neriešiteľné. V tom čase sa začali počítače stále viac a viac používať aj v civilnom sektore. Pri praktickej realizácii algoritmov

nešlo ani tak o ich existenciu, ako o ich výpočtovú zložitosť, a tým aj o efektívnosť výpočtov. Pojmom a konceptom teórie zložitosti sa venuje piata kapitola s názvom

> *Teória zložitosti alebo čo môžeme robiť, keď na výpočet nestačí energia celého vesmíru?*

Po pojme algoritmu je pojem zložitosti druhým najdôležitejším pojmom informatiky. Zložitosť chápeme v informatike v prvom rade ako výpočtovú zložitosť, presnejšie, ako množstvo práce, ktoré musí počítač vykonať, aby sa dopracoval k výsledku. Najčastejšie meriame zložitosť algoritmu ako počet vykonaných operácií alebo ako veľkosť používanej pamäti. Pokúsime sa tiež merať zložitosť problémov ako zložitosť najlepších (najrýchlejších, alebo najúspornejších pri zaobchádzaní s pamäťou) algoritmov, ktoré daný problém riešia.

Prvoradou úlohou teórie zložitosti v informatike je klasifikovať problémy (výpočtové úlohy) na ľahké a ťažké, vzhľadom na ich výpočtovú zložitosť. Vieme, že existujú ľubovoľne ťažké algoritmické problémy a poznáme tisíce úloh z praxe, pri riešení ktorých potrebujú uskutočniť aj najlepšie známe algoritmy podstatne viac operácií, ako je protónov vo vesmíre. Celá energia vesmíru, ani čas trvania vesmíru sú nedostatočné zdroje na ich riešenie. Máme vôbec nejakú šancu podniknúť niečo v tejto situácii?

Na tomto mieste začína nadobúdať konkrétnu podobu prvý div informatiky v tejto sérii prednášok. Môžeme urobiť veľa. Prečo a ako je to možné, to je pravé umenie algoritmiky. Mnohé ťažké problémy sú veľmi „citlivé“ a v nasledujúcom zmysle nestabilné. Malá zmena formulácie úlohy alebo malé oslabenie našich požiadaviek, môže spôsobiť obrovský skok (vo výpočtovej zložitosti) z fyzikálne nerealizovateľného množstva práce na výpočty, ktoré sa dajú zrealizovať v zlomku sekundy. Témou tejto kapitoly a tiež viacerých po nej nasledujúcich je, ako sa dajú dosahovať takéto efekty vďaka originálnym konceptom algoritmiky.

Zázrak nastane vtedy, keď sa nám podarí naše požiadavky znížiť v takej miere, že z praktického hľadiska o redukciu požiadaviek vlastne ani nejde, a napriek tomu dôjde k obrovskej úspore počítačovej práce.

Najmagickejšie príklady riešenia zdanlivo bezvýchodiskových situácií vznikajú pri použití náhodného riadenia algoritmov a výpočtových systémov. Dosiahnutý efekt je natoľko fascinujúci, že pôsobí ako skutočný zázrak. Preto tejto téme venujeme špeciálnu kapitolu s názvom:

> *Náhoda a jej úloha v prírode alebo náhoda ako zdroj efektív-*

nosti v algoritmike.

Myšlienka spočíva v opustení deterministického priebehu programu a dovolení algoritmu občas si hodiť mincou. V závislosti od toho, či padne (rub alebo líc), si algoritmus môže zvoliť rôzne stratégie, ako bude hľadať riešenie. Tým obetujeme absolútnu spoľahlivosť v zmysle garancie správneho výsledku, pretože niektoré postupnosti hodov mincou môžu viesť k neúspešnému výpočtu. Neúspešný výpočet je buď výpočet, ktorý nevedie k nijakému výsledku, alebo výpočet vedúci dokonca k zlému výsledku. Keď ale vieme zmenšiť pravdepodobnosť neúspešného výpočtu na jeden z miliardy, algoritmus môže byť veľmi užitočný.

Zdôraznime skutočnosť, že v praxi môžu byť randomizované algoritmy s malou pravdepodobnosťou chybného výsledku dokonca spoľahlivejšie ako ich najlepšie deterministické náprotivky. Čo tým myslíme? Teoreticky sú všetky deterministické programy správne a randomizované sa môžu mýliť. Podstata tkvie v tom, že beh deterministického programu nie je absolútne bezchybný, pretože počas výpočtu rastie úmerne s dĺžkou výpočtu aj pravdepodobnosť hardvérovej chyby. Preto môže byť rýchly randomizovaný algoritmus spoľahlivejší ako pomalý deterministický. Napríklad, ak randomizovaný program vypočíta výsledok za 10 sekúnd, potom je spoľahlivejší ako deterministický program, ktorý vypočíta výsledok za týždeň. Využitím randomizácie získame enormný nárast efektívnosti, keď akceptujeme veľmi malú stratu spoľahlivosti. Keď ale vďaka tejto len zdanlivej strate spoľahlivosti výpočtov spôsobíme skok z fyzikálne nerealizovateľného množstva práce na množstvo práce uskutočniteľné za pár sekúnd na bežnom PC, tak môžeme hovoriť o naozajstnom zázraku. Bez takýchto zázrakov informatiky by neexistovala dnešná internetová komunikácia, elektronické obchodovanie a bankovníctvo.

Okrem aplikácií náhody v informatike v tejto kapitole diskutujeme aj o principiálnej otázke – existencii pravej náhody a ukazujeme, ako sa vzťah k náhode menil v priebehu dejín vedy.

Pod názvom:

Kryptografia alebo ako spraviť zo slabiny prednosť

ponúkame v siedmej kapitole dejiny kryptografie, ktorá sa stala vedou o šifrovaní. V priebehu rozprávania sa dozvieme, ako sa kryptografia vďaka algoritmike a konceptom teórie zložitosti vyvinula z hry na šifrovanie na seriózny vednú disciplínu. Je ťažké nájsť vednú oblasť, ktorá by obsahovala také množstvo divov v zmysle výskytu prekvapujúcich zvratov a otváraní neuveriteľných možností pri hľadaní riešení.

Kryptografia je starodávna veda o tajných písmach. Jej úlohou je zašifrovať (zakódovať) zrozumiteľné texty tak, aby ich nemohol dešifrovať, a teda čítať nikto, okrem určeného adresáta. Klasická kryptografia je založená na tajných „kľúčoch“, ktoré sú spoločným tajomstvom odosielateľa a príjemcu.

Vo vývoji kryptografie informatika zohrala kľúčovú úlohu. Najprv umožnila na úrovni tvorenia pojmov po prvýkrát definovať bezpečnosť (spoľahlivosť) kryptosystémov. Kryptosystém je pre užívateľov bezpečný, keď každý program, ktorý nepozná kľúč, vyžaduje na dešifrovanie textu fyzikálne nerealizovateľné množstvo výpočtov. Na základe definície kvality kryptosystému objavili informatici také efektívne realizovateľné šifrovacie metódy, ktorých dešifrovanie bez znalosti kľúča zodpovedá ťažkým algoritmickým problémom.

Na tomto príklade vidíme, že existencia ťažkých problémov neslúži len na to, aby nám ukázala hranice našich technických možností, ale aj na to, aby nám pomohla riešiť kryptografické úlohy. Vďaka tejto myšlienke sa podarilo vyvinúť takzvané kryptosystémy s verejným (neutajeným) kľúčom. Šifrovací kľúč a aj celý šifrovací mechanizmus môže byť zverejnený v telefónnom zozname. A to vďaka tomu, že na dešifrovanie je potrebný ešte jeden tajný kľúč, ktorý pozná len právoplatný príjemca. Bez tohto tajomstva nie je schopný žiadny iný subjekt dešifrovať zašifrovaný text v realistickom čase napriek tomu, že pozná šifrovaciu metódu.

Nasledujúce dve kapitoly sa venujú možnostiam enormného miniaturizovania počítačov. Hlavnou myšlienkou je vykonávanie výpočtov na úrovni molekúl a elementárnych častíc.

Pod názvom

> *Počítanie s DNA molekulami alebo biopočítačová technológia na obzore*

predstavujeme v ôsmej kapitole biochemické technológie, ktoré by sa mohli dať využiť na riešenie ťažkých úloh. Na jednoduchom príklade ťažkej úlohy ukážeme, ako sa dajú údaje reprezentovať pomocou reťazcov DNA a ako sa dá vypočítať výsledok pomocou chemických operácií na týchto reťazcoch.

Ak sa detailnejšie pozrieme na prácu počítačov, zistíme, že nerobia nič iné, len transformujú texty (postupností symbolov) na iné texty. Počítač dostane vstupné údaje opisujúce problém vo forme postupnosti symbolov (napríklad núl a jednotiek), výstup počítača je znova len text v tvare po-

stupnosti znakov. V priebehu výpočtu sú údaje uložené v pamäti počítača ako postupnosti núl a jednotiek. Počítač „počíta“ s týmito postupnosťami.

Môže niečo také napodobniť príroda? Na reťazce DNA molekúl môžeme tiež nazerať ako na postupnosti štyroch symbolov A,T,C a G a vieme tiež, že DNA molekuly sú nositeľmi informácie rovnako, ako dáta v počítači. Rovnakým spôsobom ako počítače realizujú operácie nad symbolickou reprezentáciou údajov, umožňujú chemické operácie meniť biologické údaje. Čo dokážu počítače, zvládnu molekuly ľavou rukou a dokonca podstatne rýchlejšie.

V tejto kapitole rozoberáme možnosti biologických technológií v informatike, ich silné a slabé stránky. Táto oblasť je plná prekvapení a meniacich sa názorov, nik sa dnes neodváži formulovať predpovede o možných aplikáciách týchto technológií o 10 rokov.

Ťažko by sme hľadali okrem fyziky inú vednú disciplínu, ktorá by tak výrazne ovplyvnila náš pohľad na svet v poslednom storočí. S fyzikou sa spájajú hlboké poznatky a fascinujúce objavy. Klenot medzi diamantmi je kvantová mechanika. Význam jej objavu znesie prirovnanie k objavu ohňa v dobe kamennej. Fascinácia kvantovou teóriou spočíva v tom, že zákony správania sa elementárnych častíc odporujú našim skúsenostiam z makrosveta. Táto, na začiatku veľmi kritizovaná, a dnes uznávaná fyzikálna teória, umožňuje hypoteticky nový spôsob počítania na úrovni elementárnych častíc. Tejto téme venujeme deviatu kapitolu s názvom

> *Kvantové počítače alebo počítanie v čarovnom svete mikročastíc.*

Hneď ako vedci odhalili túto možnosť, položili si otázku, či ešte platí prvá axióma informatiky. Inými slovami: Dokážu kvantové počítače niečo, čo klasické nedokážu? Rýchlo sa našla negatívna odpoveď, takže kvantové algoritmy riešia tú istú triedu algoritmicky riešiteľných úloh, ktorú sme definovali pomocou klasických algoritmov. Vďaka tomuto poznatku stojí naša axióma informatiky na vlastných nohách ešte pevnejšie a o jej stabilite sa dnes nepochybuje. Aké sú potom ale prednosti používania kvantových počítačov? Existujú konkrétne úlohy veľkého praktického významu, ktoré sa dajú efektívne riešiť na kvantových počítačoch, zatiaľ čo doteraz najlepšie vytvorené klasické algoritmy potrebujú na ich riešenie vykonať nerealizovateľné množstvo počítačovej práce. Vďaka tomu je kvantová mechanika pre nás veľmi sľubnou počítačovou technológiou. Problém je len v tom, že zatiaľ nie sme schopní konštruovať používateľné kvantové počítače a dosiahnutie tohto cieľa je veľkou výzvou súčasného fy-

zikálneho výskumu. Vzhľadom na to, že pochopenie kvantových výpočtov vyžaduje hlbšie poznatky z matematiky, nebudeme sa usilovať do podrobností predviesť „hlavné myšlienky“ kvantovej algoritmiky. Vysvetlíme len na jednoduchom príklade, čo všetko je možné vo svete elementárnych častíc a ako sa to dá využiť na realizovanie výpočtov. Objasníme tiež, prečo je zostrojenie kvantového počítača takou náročnou úlohou a aké neuveriteľné možnosti pre bezpečnú a utajenú komunikáciu sa nám prostredníctvom kvantovej mechaniky otvárajú v kryptografii.

Desiata kapitola s názvom:

> *Ako dospieť k dobrým rozhodnutiam bez poznania budúcnosti, alebo ako prekabátiť prefíkaného protivníka*

sa vracia späť k algoritmike, vedeckému jadru informatiky.

V živote často nastávajú situácie, v ktorých by sme radi vedeli, čo nám prinesie budúcnosť. Žiaľ len zriedka do budúcnosti môžeme nahliadnuť, alebo jej vývoj odhadnúť, teda musíme rozhodnutia prijímať dnes a bez toho, aby sme poznali budúcnosť. Predstavme si napríklad lekársku pohotovostnú službu s mobilnými lekármi. Nikto nevie dopredu, odkiaľ a kedy príde tiesňové volanie. Napriek tomu sa ale môže riadiaca služba pokúšať efektívne koordinovať pohyb lekárov. Za cieľ si môže určiť minimalizáciu priemerného času čakania na pohotovostného lekára alebo celkového počtu najazdených kilometrov.

Na splnenie uvedených cieľov môže centrála vyvinúť rôzne stratégie. Napríklad určí, čo má lekár ďalej podniknúť po úspešnom zásahu. Má čakať na nasledujúce tiesňové volanie tam, kde je? Má sa vrátiť naspäť do nemocnice, alebo sa má dokonca presunúť na nejakú novú, strategicky výhodnú vyčkávaciu pozíciu? Ktorého lekára vyšle centrála pri aktuálnom tiesňovom volaní? Principiálna otázka je, či pre takéto, takzvané online problémy, vôbec existuje nejaká rozumná stratégia, ktorá je v istom zmysle úspešná bez ohľadu na to, čo nám prinesie budúcnosť.

Celú túto online úlohu si môžeme predstaviť ako hru proti nejakému prefíkanému protihráčovi. Ihneď ako spravíme a zrealizujeme nejaké rozhodnutie, protihráč nám vytvorí takú budúcnosť, pre ktorú je naše rozhodnutie čo najnepriaznivejšie. Máme v takejto hre vôbec reálnu možnosť robiť rozumné a úspešné rozhodnutia? Odpovede na túto otázku môžu byť rôzne a závisia od konkrétnych online úloh. Je fascinujúce pozorovať, že pomocou informatiky často môžeme prekabátiť aj takého silného protihráča, ktorý môže tvoriť pre nás nepriaznivú budúcnosť.

Knihu uzavrieme jedenástou kapitolou s názvom:

> *Algoritmická optimalizácia vo fyzike alebo ako môže účinkovať homeopatia.*

Na rozdiel od predchádzajúcich kapitol tu opustíme relatívne pevnú pôdu uznávaných vedeckých výsledkov a dovolíme si brať vážne zopár vizionárskych domnienok, ktoré nám otvoria novú dimenziu nazerania na život, zdravie a liečenie.

Z termodynamiky vieme, že každý fyzikálny systém sa neprestajne pokúša dosiahnuť svoj ideálny stav, ktorý nazývame optimum. Napriek tomu následkom silného vonkajšieho zaťaženia a rôznych vplyvov sa môže od svojho optima vzďaľovať. Proces návratu systému k optimu vedia fyzici veľmi dobre simulovať pomocou takzvaného Metropolisovho algoritmu.

Na začiatku osemdesiatych rokov dvadsiateho storočia vedci prekvapujúco objavili, že algoritmickú optimalizáciu v matematike a v informatike je možné modelovať ako proces optimalizácie nejakého fyzikálneho systému. Na základe tohto princípu vyvinuli heuristiku „simulovaného žíhania“, ktorá zaznamenala pri riešení ťažkých optimalizačných úloh veľké úspechy v praxi.

Dovoľme si teraz prijať predpoklad, že živé systémy (organizmy) sa správajú podobne ako „neživá“ hmota v tom zmysle, že sa nepretržite snažia dosiahnuť svoj ideálny stav. Zdravie považujme za optimálny stav a chorobu za odklon od tohto stavu. Na základe takéhoto pohľadu vytvoríme vizionársku predstavu medicíny budúcnosti, ktorá sa namiesto lokálnych korektúr a ošetrení, sústredí na povzbudenie a podporu vlastných optimalizačných schopností organizmu. Pri takomto ponímaní vyzerajú mnohé alternatívne liečebné praktiky, ako napríklad homeopatia, prirodzene a realisticky. Ukážeme dokonca, že isté experimentálne pozorovania procesu liečenia po podaní homeopatických prípravkov zodpovedajú priebehu optimalizácie opísanej Metropolisovým algoritmom.

Kniha „Sedem divov informatiky“, ktorú by sme mohli nazvať aj „Algoritmické dobrodružstvá„ sa končí jedenástou kapitolou. Koľko divov sme ukázali naozaj, nechceme teraz počítať a ani to nikomu nedoporučujeme. Výsledný počet závisí od toho, ako definujeme pojem div. A už sme sa naučili, akou náročnou prácou je tvorenie pojmov. Takže namiesto počítania divov, čitateľovi ponúkame dva doslovy.

Skúsenému čitateľovi, ktorý úspešne v plnom zdraví strávil predchádzajúce ťažké témy sa v prvom doslove snažíme sprostredkovať náš pohľad

na príspevok informatiky k všeobecnému vzdelaniu. Informatika prirodzeným spôsobom spája v jednom odbore matematicko-prírodovedné myslenie s pragmatickými postupmi inžinierskych disciplín. Toto spojenie otvára novú dimenziu, ktorá zatiaľ stredným školám chýba. Atraktívnosť informatiky je vecou vysokej miery interdisciplinarity a prepojenia teórie a experimentov.

Čo dnes na stredných školách chýba na to, aby bola informatika akceptovaná, sú učebnice a učebné materiály, ktoré by boli schopné sprostredkovať vyššie uvedené hodnoty. Dúfame, že touto knihou sa nám podarilo ukázať, ktorým smerom sa môžeme vybrať pri tvorbe takýchto učebníc.

Druhý doslov je manifestom na podporu základného výskumu. Príliš veľa politikov, podnikateľov a manažérov volá po aplikovanom výskume, ktorý prináša zisk v krátkom čase. Miesto generovania nových poznatkov v základnom výskume sa má výrazne uprednostňovať transformácia získaného poznania do ziskových produktov. Štátna politika financovania vedy sa má sústrediť namiesto tvorby nových poznatkov na spotrebu poznatkov v mene populisticky proklamovaného, zdanlivého rastu životnej úrovne. V tomto manifeste vysvetľujeme, prečo je tento prístup k financovaniu vedy rovnako životu nebezpečný, ako bolo pre ľudí v dobe kamennej krátkodobé prejedenie a nasýtenie sa namiesto vytvárania zásob na zimu.

1.5 Zhrnutie

Pre vznik a vývoj vedných disciplín je určujúce tvorenie pojmov. Definovaním pojmu algoritmus dali vedci pojmu metóda presný význam a položili tým základný kameň novej vedy – informatiky. Vďaka tejto definícii sme získali jasnú hraničnú čiaru medzi automaticky (algoritmicky) riešiteľnými a automaticky neriešiteľnými úlohami. Potom, ako sa podarilo úspešne roztriediť mnohé úlohy na algoritmicky neriešiteľné a algoritmicky riešiteľné, prišiel pojem výpočtovej zložitosti, ktorý dodnes dominuje základnému výskumu v informatike. Tento pojem umožňuje skúmať hranicu medzi praktickou riešiteľnosťou a praktickou neriešiteľnosťou, kryptografii poskytol základ pre pojem bezpečného šifrovacieho systému, a tým umožnil objavenie moderných kryptosystémov s verejným kľúčom. Hlavne umožnil skúmať a porovnávať výpočtovú silu determinizmu, nedeterminizmu, náhodného riadenia a kvantových algoritmov. Takto prispela a prispieva informatika nielen k chápaniu všeobecných kategórií vedy, akými sú:

determinizmus, nedeterminizmus, náhoda, informácia, pravda, nepravda, zložitosť, jazyk, dôkaz, poznatok, komunikácia, algoritmus, simulácia, atď.,

ale dáva viacerým z týchto kategórií aj nový význam. Divotvorné objavy informatiky sú spojené najmä s úsilím riešiť ťažké úlohy. Vznikajú pritom „divy“, ktorým je venovaná táto kniha.

Vzorové riešenia k vybraným úlohám

Úloha 1.3 V pravdivostnej tabuľke na obrázku obr. 1.5 sú len tri prípustné (možné) situácie a to S_1, S_2 a S_8. Spytujeme sa, ktoré implikácie platia. Na zodpovedanie tejto otázky používame nasledovné pravidlo:

Ak vo všetkých možných situáciách, v ktorých platí X, platí aj Y, tak platí $X \Rightarrow Y$. Implikácia $X \Rightarrow Y$ neplatí, ak existuje situácia, v ktorej X platí, ale Y neplatí.

Skúmajme najprv platnosť implikácie $A \Rightarrow B$. Jediné možné situácie, v ktorých A platí, sú S_1 a S_2. V týchto situáciách platí aj B. Teda usúdime, že platí $A \Rightarrow B$.

Teraz sa pozrime na $B \Rightarrow A$. B platí v S_1 a S_3 a v tých situáciách platí aj A. Takže platí aj $B \Rightarrow A$.
Skúmajme, či platí implikácia $A \Rightarrow C$. Z možných situácií, A platí v S_1 a S_2. V situácii S_2 ale C neplatí. Teda implikácia $A \Rightarrow C$ neplatí.
Naopak, ale platí implikácia $C \Rightarrow A$, pretože C platí len v S_1 a tam platí aj A. Postupujúc týmto spôsobom prídeme k tomu, že platia implikácie $A \Rightarrow B$, $B \Rightarrow A$, $C \Rightarrow A$ a $C \Rightarrow B$ a že neplatia implikácie $A \Rightarrow C$ a $C \Rightarrow B$. Implikácie $A \Rightarrow A$, $B \Rightarrow B$ a $C \Rightarrow C$ platia vždy nezávisle od toho, ktoré situácie sú možné.

Úloha 1.6 Načrtnime najprv pravdivostnú tabuľku pre C a D a skúmajme, kedy platí $C \Rightarrow D$.

	C	D	$C \Rightarrow D$
S_1	platí	platí	
S_2	platí	neplatí	vylúčené
S_3	neplatí	platí	
S_4	neplatí	neplatí	

Z tabuľky vidíme, že sú možné situácie S_1, S_3 a S_4. Čo znamená, vziať do úvahy dodatočnú informáciu, že pri miešaní nevznikla zelená farba? Nič iné, len to, že D neplatí, a teda, že platí $\overline{D}$. To vylučuje situácie S_1 a S_3, v ktorých platí D. Ostáva len jedna jediná možná situácia, a tou je S_4. V situácii S_4 platia $\overline{D}$ a $\overline{C}$, teda platí aj $\overline{D} \Rightarrow \overline{C}$. Keďže vieme, že miešaním nevznikla zelená farba ($\overline{D}$ platí), môžeme usúdiť, že sme nezmiešali žltú a modrú farbu ($\overline{C}$ platí).

Úloha 1.8 Uvažujme nad dvomi tvrdeniami. Tvrdenie A znamená, že „x^2 *je párne*“ a tvrdenie B znamená, že „x *je párne*“. Platnosť A je daná. Naším cieľom je dokázať, že platí tvrdenie B. V nepriamom dôkaze musíme začať s $\overline{B}$. Tvrdenie $\overline{B}$ znamená, že „x *je nepárne*“. Podľa definície nepárnych čísel vieme, že x môžeme vyjadriť ako:

$$x = 2i + 1,$$

pre nejaké prirodzené číslo i. Teda platí $x = 2i + 1$, čo označíme ako tvrdenie A_1. Týmto sme ukázali platnosť implikácie $\overline{B} \Rightarrow A_1$. Umocnením x na druhú dostaneme z A_1 nasledujúce tvrdenie A_2:

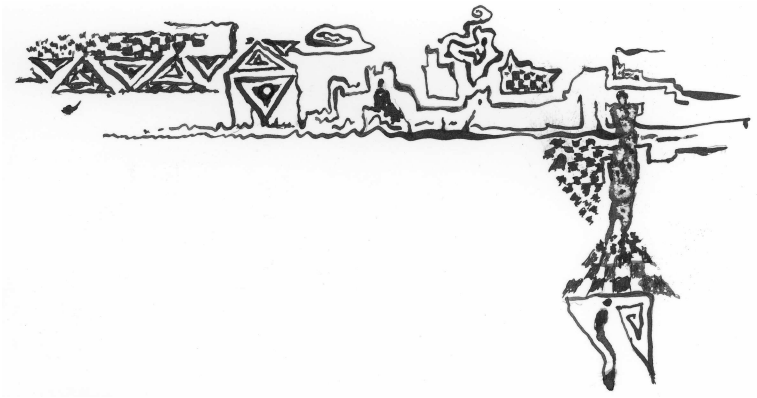
$$x^2 = (2i + 1)^2 = 4i^2 + 4i + 1 = 2(2i^2 + 2i) + 1 = 2m + 1.$$

Vidíme, že pre $m = 2i^2 + 2i$ je $x^2 = 2m + 1$ (x^2 je nepárne, pretože x^2 môžeme vyjadriť ako dvakrát nejaké prirodzené číslo plus 1). Dostali sme tvrdenie $\overline{A}$, že x^2 je nepárne. Máme nasledujúcu postupnosť implikácií:

$$\overline{B} \Rightarrow A_1 \Rightarrow A_2 \Rightarrow \overline{A}$$

$$x \text{ je nepárne} \Rightarrow x = 2i + 1 \Rightarrow x^2 = 2m + 1 \Rightarrow \ x^2 \text{ je nepárne.}$$

Vieme, že x^2 je párne, teda, že neplatí $\overline{A}$. Vďaka tomu môžeme podľa schémy nepriameho dôkazu usúdiť, že nemôže platiť $\overline{B}$. Preto platí B a naším cieľom bolo dokázať platnosť B.



Dokonalosť pozostáva z maličkostí, a pritom dokonalosť nie je vôbec maličkosť.

Michelangelo Buonarotti

Kapitola 2

Algoritmika alebo čo má spoločné programovanie a pečenie koláča?

2.1 Čo sa tu dozvieme?

Cieľom tejto kapitoly ešte nie je predstaviť niektorý z divov informatiky. Nemôžeme čítať a vychutnávať Shakespeara alebo Dostojevského v origináli bez toho, aby sme neabsolvovali namáhavú cestu učenia sa anglického a ruského jazyka. Rovnako nemôžeme pochopiť informatiku a obdivovať jej myšlienky a objavy bez toho, aby sme sa naučili ovládať aspoň elementárne základy jej odborného jazyka.

Ako sme sa už usilovali vysvetliť v predchádzajúcej kapitole, centrálnym pojmom informatiky je algoritmus. Naším cieľom v tejto kapitole ale nie je absolvovanie náročnej cesty výučby informatickej terminológie, ktorá je založená do značnej miery na ovládaní jazyka matematiky. Skôr sa pokúsime sprostredkovať hravým spôsobom, bez použitia matematického aparátu, intuitívne, a napriek tomu pomerne presne, čo sú a čo nie sú

algoritmy. Začneme všeobecne známou činnosťou – pečením koláča a budeme rozmýšľať a diskutovať o tom, do akej miery a za akých okolností môžeme recepty považovať za algoritmy.

Po tomto úvode prejdeme priamo k počítačom a nazrieme na programovanie, ako na používanie jazyka na komunikáciu so strojmi. Takto predstavíme programy ako reprezentáciu algoritmov zrozumiteľnú pre počítače. Na konci tejto kapitoly budeme nielen chápať, čo znamená programovať, ale budeme dokonca schopní samostatne písať jednoduché programy a získame jasnú predstavu o tom, čo sa odohráva v počítači pri vykonávaní jednotlivých inštrukcií (počítačových príkazov).

Popri tom sa aj naučíme, čo sú algoritmické úlohy (problémy), že našou úlohou je vyvinúť algoritmy, ktoré v každej z potenciálne nekonečne mnohých situácií pracujú správne a očakávaný výsledok vypočítajú v konečnom čase. Takto si postavíme most k tretej kapitole, v ktorej chceme ukázať, nakoľko je pre informatiku dôležité hlboké pochopenie pojmu nekonečna.

2.2 Algoritmické pečenie

V prvej kapitole sme už získali istú predstavu o tom, čo sa môže skrývať pod pojmami algoritmus a metóda. Naša predstava je približne takáto:

> *Algoritmus je dobre zrozumiteľný opis činnosti, ktorý nás privedie k stanovenému cieľu.*

Inými slovami, algoritmus (metóda) nám poskytuje jednoznačne interpretovateľné pokyny, ako dospieť krok za krokom k vytýčenému cieľu.
To je podobné ako pri receptoch na varenie a pečenie. Recept určuje presne, čo a v akom poradí treba vykonávať a nám neostáva nič iné, ako vykonávať opísanú činnosť presne krok za krokom.

> *Do akej miery môžeme pokladať recept za algoritmus?*

Odpoveď na túto otázku nie je jednoduchá, a preto jej venujeme viac priestoru. Pri hľadaní správnej odpovede pochopíme presnejšie, čo sa v skutočnosti skrýva za slovom algoritmus.

Pozrime si najprv nasledujúci recept na pečenie marhuľového koláča priemeru 26 cm.

Suroviny:
3 vajíčkové bielka
1 štipka soli
6 polievkových lyžíc vody
100g kryštálového cukru
3 vajíčkové žĺtka
1 čajová lyžička nastrúhanej citrónovej kôry
150g múky
1/2 čajovej lyžičky prášku na pečenie
400g odkôstkovaných a ošúpaných marhúľ

Recept:

1. Uložte pergamenový papier do formy na pečenie!
2. Vyhrejte rúru na 180 °C!
3. Zohrejte 6 lyžičiek vody!
4. Zmiešajte tri bielka, štipku soli a 6 lyžičiek horúcej vody a ušľahajte z nich pevný sneh!
5. Pridajte postupne 100g cukru a tri žĺtka. Miešajte ich tak dlho, pokiaľ nevznikne pevná krémová masa!
6. Pridajte do krému 1 čajovú lyžičku nastrúhanej citrónovej kôry a premiešajte!
7. Zmiešajte 150g múky a polovicu čajovej lyžičky prášku na pečenie a pridajte k vyhotovenej mase. Premiešajte masu opatrne šľahačom!
8. Naplňte zmiešanou masou formu na pečenie!
9. Umiestnite ozdobne ošúpané polovice marhúľ na cesto!
10. Pečte koláč pri 160 °C 25 až 30 minút, pokiaľ nenadobudne svetlohnedú farbu!
11. Vyberte koláč z rúry a nechajte ho vychladnúť!

Recept je napísaný a my sa spytujeme: Je skutočne každý schopný podľa tohto receptu upiecť koláč? Pravdepodobne správna odpoveď je, že úspech predsa len bude do značnej miery závisieť od kuchárových skúseností a vedomostí.

Dospeli sme k bodu, keď je rozumné presnejšie sformulovať našu prvú požiadavku na definíciu pojmu algoritmus:

> *Algoritmy musia poskytovať taký presný opis činnosti, že podľa tohto opisu môže každý vykonávať úspešne danú činnosť, aj keď vôbec nerozumie dôvody, prečo vedie presné vykonávanie pokynov algoritmu k vytýčenému cieľu. Pritom opis musí byť natoľko jednoznačný, že nemôže dôjsť k rôznym interpretáciám jednotlivých pokynov (príkazov) algoritmu. Nezávisle od toho, kto algoritmus vykonáva, príslušná činnosť a aj jej výsledok musia byť tie isté. To znamená, že každý používateľ algoritmu musí dospieť k tomu istému výsledku.*

Teraz by sme mohli začať dlhú diskusiu o tom, ktoré z uvedených 11 krokov (príkazov) nášho receptu možno pokladať za zrozumiteľné a jednoznačne interpretovateľné. Napríklad by sme sa mohli začať pýtať:

- Čo znamená **ušľahať pevný sneh** (krok 4)?
- Čo znamená **opatrne premiešať** (krok 7)?
- Čo znamená **ozdobne uložiť** (krok 9)?
- Čo znamená **dosiahnuť svetlohnedú farbu** (krok 10)?

Skúsená kuchárka by mohla povedať: „Všetko je úplne jasné, jednoduchšie to už ani nemožno opísať.“ Ale niekto, kto sa prvýkrát v živote chystá piecť koláč, sa ešte stále môže cítiť neistý a potrebuje radu a pomoc. A to napriek tomu, že náš recept je opísaný jednoduchšie ako vo väčšine kuchárskych kníh. Čo si myslíte o typických pokynoch (príkazoch) bežných receptov, ako napríklad:

- Zmiešajte **bez otáľania mierne ochladenú** želatinu a kyslé mlieko a **dobre** ich premiešajte!

Samozrejme nemôžeme sa uspokojiť s tým, že náš recept budú považovať za algoritmus len skúsení kuchári a ostatní mu nebudú dostatočne rozumieť. Dá sa prepísať recept tak, aby ho vnímali všetci ako algoritmus? Už vieme, že algoritmus je postupnosť takých príkazov, ktoré musí byť schopný každý správne zrealizovať.
To znamená:

> *Najskôr sa musíme dohodnúť na nejakom zozname činností (operácií), ktoré dokáže s istotou vykonávať každý záujemca o pečenie a varenie.*

Takýto zoznam musí obsahovať v prvom rade nasledujúce činnosti (príkazy), ktoré sú také jednoduché, že ich dokáže zvládnuť aj na tento účel

skonštruovaný robot, ktorý nemá tušenia o kuchárskom umení a nie je schopný improvizácie.

- `Nalej x lyžičiek vody (alebo inej tekutiny) do nádoby!`
- `Rozdeľ vajíčko na žĺtko a bielko!`
- `Zohrej rúru na x °C`
- `Peč v rúre y minút pri teplote x °C!`
- `Odváž x gramov múky a daj ju do nádoby!`
- `Nalej x litrov mlieka do konvice!`
- `Var y minút!`
- `Miešaj so šľahačom x minút!`
- `Zmiešaj obsah dvoch nádob.`

Určite vám príde na um ešte množstvo ďalších aktivít, ktoré môžeme pokladať za také jednoduché, že takmer každý, kto chce piecť a variť, je schopný ich vykonávať bez pomoci. Našou nasledujúcou úlohou je prepísať aspoň časť nášho receptu tak, aby recept pozostával len z jednoduchých činností.

Vyskúšajme to s krokom 4 nášho receptu. Tento krok prepíšeme na postupnosť siedmich jednoduchších krokov.

```
4.1 Daj tri žĺtka do nádoby G!
4.2 Daj 1g soli do nádoby G!
4.3 Daj 6 lyžičiek vody do hrnca T!
4.4 Zohrej vodu v hrnci T na 60 °C.
4.5 Prelej vodu z hrnca T do nádoby G!
```

V tomto okamihu ale nie je jasné, ako zrealizovať nasledujúci príkaz: „Ušľahaj z nich pevný sneh!“. Našou úlohou je miešať dovtedy, pokiaľ sneh z bielok nebude pevný. Jedna z možností je, na základe skúseností odhadnúť čas miešania. Ak trvá približne 2 minúty, pokým je sneh pevný. Potom môžeme použiť nasledovný príkaz.

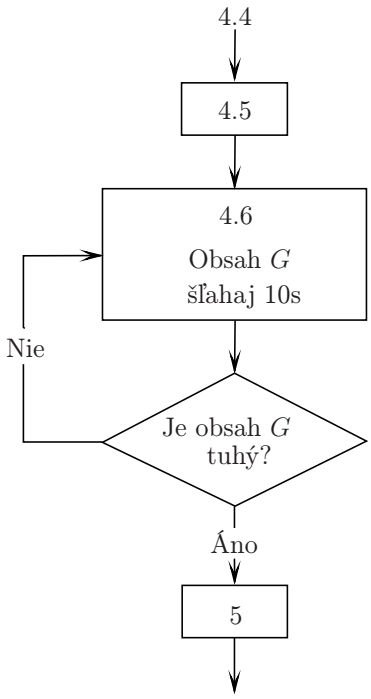
```
4.6 Miešaj obsah nádoby G dve minúty!
```

Takýto príkaz má ale aj svoje riziká. Čas vyhotovenia pevného snehu závisí od toho, ako rýchlo a s akými pomocnými prostriedkami kuchár

mieša. Asi by nám bolo milšie prestať miešať približne vtedy, keď obsah nádoby dostatočne spevnel. Čo na to potrebujeme? Okrem schopnosti vykonávať isté činnosti musíme byť schopní vykonávať aj testy a v závislosti od výsledku testu prijať rozhodnutia, ako v práci ďalej pokračovať. Testovanie nám umožní rozpoznať okamih, keď sa sneh stáva pevným. Ak pri testovaní zistíme, že sneh nie je dostatočne pevný, tak budeme ešte istý čas pokračovať v šľahaní, a potom znova testovať. Ak pri testovaní zistíme, že sneh je dostatočne pevný, ukončíme prácu na realizácii kroku 4 a pokračujeme s prácou na uskutočnení príkazu 5.

Ako sa dá tento postup zapísať ako postupnosť jednoduchých príkazov? Napríklad nasledujúco:

```
4.6 Miešaj obsah nádoby G po dobu 10s!

4.7 Otestuj, či je obsah nádoby G pevný!
    Ak ÁNO, pokračuj krokom 5!
    Ak NIE, pokračuj krokom 4.6!
```



obr. 2.1

Pri tomto postupe sa k miešaniu v 4.6 vraciame tak dlho, pokiaľ nedosiahneme vytýčený stav snehovej masy. V odbornej terminológii informatiky

nazývame časť opisu 4.6 a 4.7 **cyklom**, v ktorom sa opakuje činnosť 4.6 tak dlho, pokiaľ nie je splnená podmienka 4.7. Pre lepšiu predstavu často používame grafickú reprezentáciu, ako na obrázku 2.1. Grafickú reprezentáciu nazývame **diagram**.

Dá sa ale test 4.6 tak ľahko realizovať? Rovnako ako v prípade jednoduchých akcií, sa musíme najprv dohodnúť na starostlivo zvolenom zozname jednoduchých testov. Test 4.6 sa dá napríklad jednoducho zrealizovať tak, že nožom sa pokúsime bielkovú masu rozrezať, ak sa to podarí, sneh už je pevný. Za jednoduché testy môžeme považovať napríklad nasledovné overovania:

- `Over, či tekutina v hrnci má aspoň x °C!`
- `Over, či v mase v nádobe bude „stáť lyžička“!`
- `Váži obsah nádoby práve x g?`

Úloha 2.1 Vyberte z vašej kuchárskej knihy recept na zhotovenie vášho obľúbeného jedla. Zostavte vlastný zoznam jednoduchých činností a testov a prepíšte Váš recept tak, aby absolvoval len jednoduché aktivity z Vášho zoznamu!

Úloha 2.2 Vašou úlohou je zohriať 1l vody na 90 °C. K dispozícii máte nasledovné operácie:

- `Postav hrniec T na x sekúnd na horúcu platňu a potom ho z platne odstav!`
- `Nalej x litrov vody do hrnca T!`

K dispozícii máte tiež nasledujúci test:

- `Má voda v hrnci aspoň x °C?`

Napíšte algoritmus na zohriatie 1 litra vody na 90 °C len pomocou vyššie uvedeného testu a príkazov! Pritom musíte zabezpečiť, aby voda po dosiahnutí 90 °C neostala stáť na platničke dlhšie ako 15 sekúnd.

Či veríte alebo nie, ak ste vyriešili tieto dve úlohy, tak ste už trocha programovali. To najdôležitejšie, čo sme sa naučili pri pečení marhuľového koláča je, že o algoritmoch nemôžeme hovoriť, pokiaľ najskôr jednoznačne neurčíme, čo sú základné kamene, z ktorých môžeme algoritmy vytvárať. Základnými stavebnými prvkami algoritmov sú na jednej strane jednoduché činnosti, ktoré je každý bezpochyby schopný zrealizovať a na druhej strane jednoduché testy, ktoré vie uskutočniť takisto každý.

2.3 A ako je to s algoritmami pre počítač?

V tejto časti bude naším cieľom upresniť naše predstavy o tom, čo je algoritmus prostredníctvom skúmania podobností a rozdielov medzi algoritmickým varením a algoritmickým počítaním na počítači.

Presne tak, ako pri varení, musíme sa aj pri počítačových algoritmoch najprv dohodnúť na nejakom zozname základných činností (operácií), ktoré počítače vedia vykonávať. Počítače nemajú inteligenciu a ani schopnosť improvizovať. To značne zjednodušuje našu úlohu, pretože vďaka tomu musí byť reč počítačov veľmi jednoduchá. Nikto nepochybuje o tom, že počítače môžu sčítavať, odčítavať, násobiť a deliť čísla a tiež čísla navzájom porovnávať. V prvom prípade zvykneme v informatickej terminológii hovoriť o aritmetických operáciách s číslami, ktoré ovláda v podstate každá kalkulačka. Tieto jednoduché operácie spolu so schopnosťou čítať vstupné údaje a vypisovať výsledky postačujú na to, aby sme každý algoritmus dokázali zapísať ako postupnosť takýchto operácií (akcií).

Pozorujeme teda, že kuchárske algoritmy a aj počítačové algoritmy nie sú ničím iným, ako postupnosťami jednoduchých úkonov (operácií). Medzi receptami a algoritmami v informatike existuje ale jeden podstatný rozdiel. Recepty ako algoritmy pečenia majú ako vstup suroviny a výsledkom je koláč. Jediný ich možný vstup je presný zoznam surovín potrebných na pečenie daného koláča. Pri algoritmických problémoch je to inak. Už vieme, že algoritmický problém v typickom prípade pozostáva z nekonečného množstva **prípadov** daného **problému**. Príklad algoritmického problému je riešenie kvadratických rovníc v tvare:

$$ax^2 + bx + c = 0.$$

Vstupom sú tri čísla a, b a c, ktoré jednoznačne opisujú danú kvadratickú rovnicu, a ktoré predstavujú jednotlivé vstupy pre spracovanie kvadratickej rovnice. Úlohou je nájsť všetky také čísla x, ktoré túto rovnicu spĺňajú.

Konkrétnym prípadom tohto problému je napríklad nasledujúca kvadratická rovnica:

$$x^2 - 5x + 6 = 0.$$

Máme $a = 1, b = -5$ a $c = 6$. Riešeniami tejto rovnice sú $x_1 = 2$ a $x_2 = 3$. Dosadením môžeme ľahko overiť, že:

$$2^2 - 5 \cdot 2 + 6 = 4 - 10 + 6 = 0$$

$$3^2 - 5 \cdot 3 + 6 = 9 - 15 + 6 = 0$$

a teda, že x_1 a x_2 sú skutočne riešeniami kvadratickej rovnice $x^2-5x+6 = 0$. Pretože existuje nekonečne veľa čísel, máme aj nekonečne veľa možností, ako zvoliť koeficienty kvadratickej rovnice a, b a c. Od algoritmov na riešenie kvadratických rovníc požadujeme, aby vedeli vypočítať riešenie pre každé a, b a c, teda pre každú kvadratickú rovnicu.

Dopracovali sme sa k druhej základnej požiadavke na formuláciu definície **algoritmu**.

> *Algoritmus na riešenie výpočtovej úlohy (problému) musí zaručovať korektné spracovanie každého možného prípadu problému. Korektné spracovanie znamená, že algoritmus pre každý vstup vypočíta v konečnom čase správny výsledok.*

Predstavme si teraz nejaký algoritmus na riešenie kvadratických rovníc. Z matematiky poznáme nasledovné vzorce na výpočet riešení:

$$\begin{aligned} x_1 &= \frac{-b + \sqrt{b^2 - 4ac}}{2a}, \\ x_2 &= \frac{-b - \sqrt{b^2 - 4ac}}{2a}, \end{aligned}$$

ak $b^2 - 4ac \geq 0$. Ak $b^2 - 4ac < 0$, neexistuje nijaké reálne[1] riešenie danej rovnice. Tieto vzorce poskytujú nasledovnú všeobecnú metódu na riešenie kvadratických rovníc.

Vstup: Čísla a, b a c reprezentujúce kvadratickú rovnicu $ax^2+bx+c = 0$.

Krok 1: Vypočítaj hodnotu $b^2 - 4ac$.

Krok 2: Ak $b^2 - 4ac \geq 0$, tak vypočítaj

$$x_1 = \frac{-b + \sqrt{b^2 - 4ac}}{2a},$$

$$x_2 = \frac{-b - \sqrt{b^2 - 4ac}}{2a}.$$

Krok 3: Ak $b^2 - 4ac < 0$, napíš „Neexistuje reálne riešenie“.

Uverme teraz matematikom, že táto metóda skutočne funguje. Na to, aby sme ju vedeli prepísať do formy počítačového algoritmu to aj tak nepotrebujeme vedieť.

[1] Dôvodom je, že nemôžeme odmocniť záporné číslo.

V skutočnosti chceme napísať túto metódu ako program. Pod pojmom **program** rozumieme *postupnosť počítačových inštrukcií*, ktoré sú napísané v tvare, ktorý je pre počítač zrozumiteľný. Medzi pojmami „program“ a „algoritmus“ existujú dva principiálne rozdiely.

1. Program nemusí byť reprezentáciou nejakého algoritmu. Program môže byť aj nezmyselná postupnosť inštrukcií, ktorá nevedie k žiadnemu cieľu.

2. Algoritmus nemusí byť vždy zapísaný vo formálnom jazyku počítača, pod ktorým rozumieme nejaký programovací jazyk. Algoritmus môžeme opísať aj v prirodzenom jazyku alebo v metajazyku matematiky. Napríklad pokyny „vynásob a a b“ alebo „vypočítaj $\sqrt{c}$“ sú prípustné pri opise algoritmu, zatiaľ čo v prípade programu musia byť tieto pokyny zapísané vo formalizme daného programovacieho jazyka.

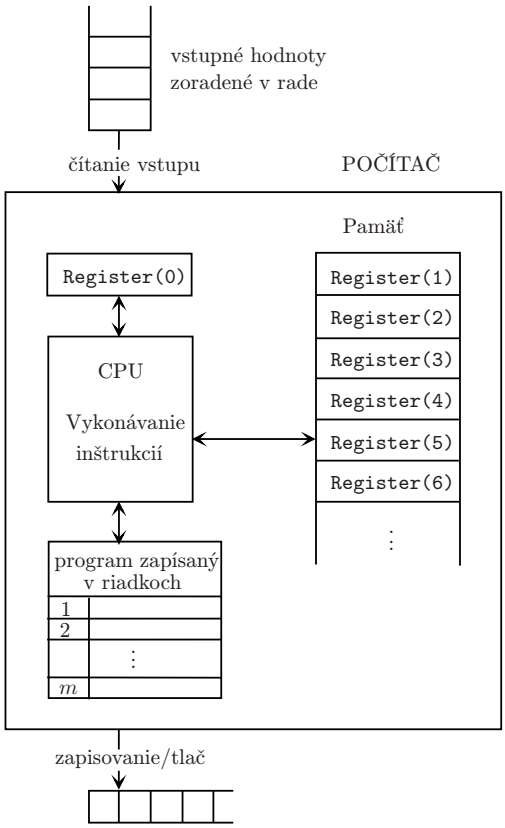
Poznamenajme, že prvý rozdiel je kľúčový a poukazuje na to, že algoritmus vždy spájame s výpočtovou úlohou, ktorú rieši. Program nemusí byť nevyhnutne spojený s riešením nejakého problému, ide len o postupnosť korektne zapísaných počítačových inštrukcií. Druhý rozdiel je len našou dohodou, že algoritmy smieme zapisovať jednoduchšie, aj bez použitia formálnych jazykov. V podstate ale vyžadujeme, aby sa dal každý algoritmus formálne presne napísať vo forme programu.

Pod **programovaním**[2] rozumieme *činnosť, pomocou ktorej prepisujeme algoritmy do formy programov.* Aby sme pochopili, ako pracuje počítač, a aby sme tiež videli, ako možno dosiahnuť zložité spracovanie informácií pomocou postupnosti veľmi jednoduchých operácií, budeme teraz trochu programovať.

Tak ako pri pečení a varení, začneme zostavovaním listiny obsahujúcej všetky povolené základné operácie (činnosti). Operácie budeme zapisovať v tvare nášho programovacieho jazyka „NÁZORNÝ“, ktorý sme vyvinuli kvôli názornej prezentácii. Vysvetľujúc význam jednotlivých inštrukcií, ukážeme, ako možno jednoducho modelovať počítač a čo presne sa v tomto počítačovom modeli odohráva pri vykonávaní jednotlivých inštrukcií.

Počítač si predstavíme do istej miery idealizovane tak, ako je nakreslený na obrázku 2.2.

[2] V širšom zmysle môžeme pod programovaním rozumieť nielen implementáciu algoritmu, ale i vývoj algoritmu na riešenie úlohy spolu s jeho implementáciou v danom programovacom jazyku.



obr. 2.2

Počítač pozostáva z nasledujúcich častí:

- **Pamäť**, ktorá pozostáva z veľkého počtu pamäťových jednotiek. Tieto pamäťové jednotky nazývame **registre**. Registre sú očíslované kladnými celými číslami (obr. 2.2). Každý register môže byť použitý na zapamätanie jedného čísla. Na začiatku výpočtu obsahujú všetky registre číslo 0. Poradové číslo registra nazývame jeho **adresou**. Napríklad číslo 112 je adresou registra `Register(112)`. Môžeme si to predstaviť ako ulicu s postupne očíslovanými domami len na jednej strane.

- Špeciálny register `Register(0)`, ktorý obsahuje číslo riadku programu, ktorý sa práve spracováva, alebo sa má práve začať spracovávať.

- Špeciálna pamäť na zapamätanie programu. Program pozostáva

z riadkov a každý riadok obsahuje práve jednu počítačovú inštrukciu.

- Centrálna procesorová jednotka, nazývaná CPU. CPU je prepojená komunikačnými kanálmi so všetkými časťami počítača. CPU pracuje v takzvaných CPU cykloch. Na začiatku cyklu si najprv prečíta aktuálny riadok programu, ktorého poradové číslo je v registri `Register(0)`. Potom si nechá poslať údaje (pamäťové obsahy) tých registrov, ktoré sú argumentmi v danej operácii. Na zaslaných údajoch vykoná danú operáciu a uloží výsledok do vyznačeného registra. Nakoniec zmení obsah registra `Register(0)` tak, aby obsahoval číslo toho riadku programu, ktorý sa má vykonať v nasledujúcom cykle.

Okrem toho je počítač spojený so svojím okolím. Vstupné údaje (čísla) čakajú v rade a počítač si môže vždy prečítať prvý údaj (číslo) v rade a zapamätať ho v nejakom registri. Počítač má tiež takzvanú výstupnú pásku, na ktorú môže vypisovať výsledky.

Pozrime sa ešte raz na podobnosť s kuchynskými receptami. Hardware počítača je kuchyňa. Registre pamäte sú nádoby ľubovoľného druhu: hrnce, misky, poháre, taniere, atď. Každá nádoba má svoje jednoznačné meno, rovnako ako majú registre svoje adresy. Takže je vždy jasné, o ktorej nádobe sa hovorí. Pamäť s programom je stránka papiera, na ktorom je napísaný recept. CPU sme my, alebo kuchynský robot so všetkými zariadeniami a nástrojmi, ako rúra, šľahač, mikrovlnka, atď., ktoré sú k dispozícii. Obsah registra `Register(0)` je pre nás aktuálna poznámka o tom, kde sa práve pri vykonávaní receptu nachádzame. Vstupy ležia v chladničke, mrazničke, alebo v špajze. V bežnom prípade síce suroviny nie sú usporiadané do radu ako naše vstupy. Ale nič nám nebráni pred varením povyberať všetky suroviny a postaviť ich do radu podľa poradia, v ktorom ich budeme potrebovať. Výstup nezapíšeme na pásku, ale položíme na jedálenský stôl.

Pri pečení a varení sme sa naučili, že prvým a zároveň hlavným krokom k špecifikácii pojmu algoritmus je dohodnúť sa na zozname **realizovateľných** inštrukcií (pokynov, činností, operácií). O ich vykonateľnosti musíme byť všetci presvedčení. V nasledujúcom budeme zo všetkých uvedených synoným uprednostňovať pojem **inštrukcie**.

Počítačové inštrukcie opíšeme radšej v prirodzenom jazyku ako v jazyku počítača v takzvanom počítačovom kóde. Začnime s operáciami čítania.

(1) `Načítaj do Register(`n`).`

Realizovať túto operáciu znamená zapamätať si v n-tom registri (`Register(`n`)`) prvé číslo zo vstupného radu. Pritom sa toto číslo zo vstupného radu vymaže a na prvú pozíciu v rade sa dostane doteraz druhé číslo vstupného radu.

Príklad 2.1 V rade čakajú tri čísla 114, -67 a 1. (pozri obr 2.3). V pamäti obsahujú všetky registre číslo 0. `Register(0)` obsahuje číslo 3. Teraz ideme realizovať nasledovnú inštrukciu:

`Načítaj do Register(3)`

z tretieho riadka programu. Po jej zrealizovaní obsahuje `Register(3)` číslo 114, ktoré stálo prvé vo vstupnom rade. Vo vstupnom rade čakajú ešte -67 a 1. Obsah registra `Register(0)` sa zvýši o 1 na 4, pretože po vykonaní inštrukcie tretieho riadka má počítač pokračovať vykonávaním inštrukcie v nasledujúcom, štvrtom riadku.

Priebeh vykonávania tejto inštrukcie je znázornený na obrázku obr. 2.3. Na tomto obrázku neznázorňujeme celý počítač, ale zameriame sa len na pamäť a vstupný rad, ktorých obsahy sa menia v priebehu vykonávania inštrukcie čítania. □

Nasledujúca inštrukcia nám umožňuje uložiť do registrov konkrétne čísla bez toho, aby sme ich museli čítať zo vstupného radu.

(2) `Register(`n`)` $\leftarrow k$

Tento príkaz požaduje, aby sme si v n-tom registri zapamätali (do n-tého registra uložili) číslo k. Pri realizácii tejto inštrukcie sa pôvodný obsah n-tého registra vymaže. Vstupný rad sa nemení.

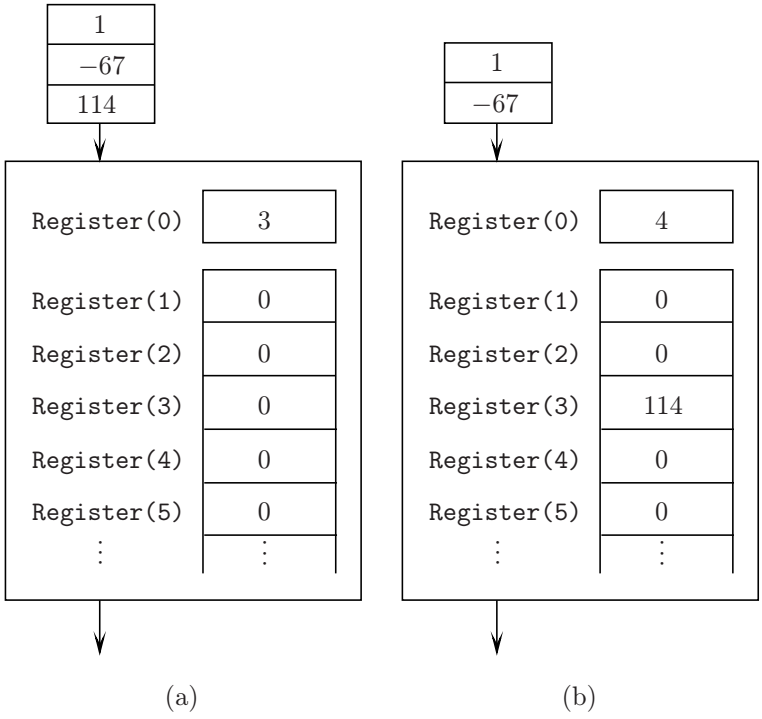
Príklad 2.2 Register `Register(50)` obsahuje číslo 100. Po vykonaní inštrukcie:

`Register(50)` $\leftarrow 22$

obsahuje `Register(50)` číslo 22. Bývalý obsah 50-teho registra, hodnota 100, sa vymaže bez toho, aby sa niekde inde zapamätal. Jeho pôvodný obsah sa definitívne stratí.

Ak je nasledujúca inštrukcia:

`Načítaj do Register(50)`



obr. 2.3

a ako prvé je vo vstupnom rade číslo 7, tak vykonaním tejto inštrukcie prepíšeme číslo 22 v 50-tom registri na číslo 7. □

Úloha 2.3 Vo vstupnom rade sú čísla 11, 12, a 13. Obsah registra `Register(0)` je 1. `Register(2)` obsahuje číslo 1117 a `Register(3)` obsahuje číslo 21. Všetky ostatné registre obsahujú číslo 0.

a) Načrtnite túto situáciu (stav počítača) podobne ako na obrázku obr. 2.3.
b) Vykonajte nasledujúci program:

```
1 Načítaj do Register(1)
2 Register(2) ← 100
3 Načítaj do Register(3)
4 Načítaj do Register(2)
```

Vypíšte obsahy všetkých registrov a vstupného radu po vykonaní jednotlivých inštrukcií.

Teraz predstavíme aritmetické operácie, ktoré je schopný vykonávať náš počítačový model.

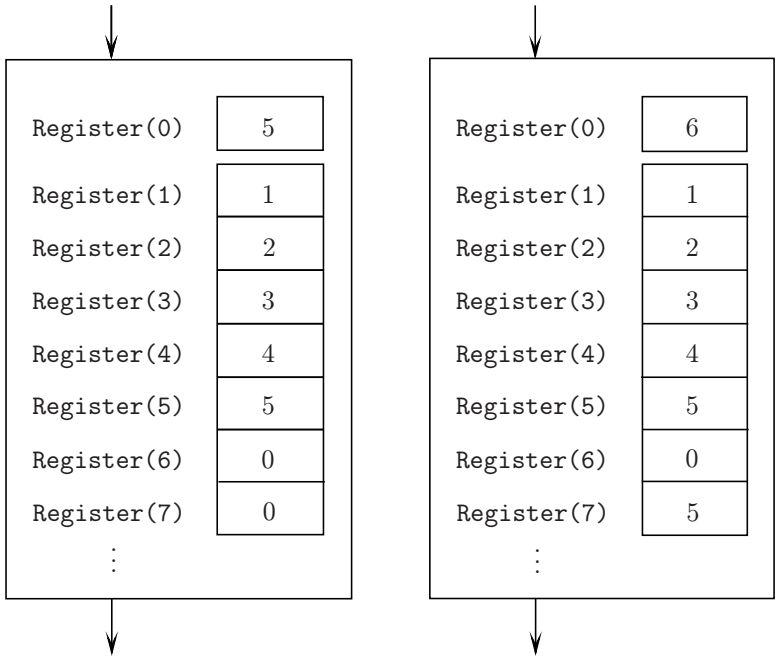
(3) `Register(`n`)` $\leftarrow$ `Register(`j`) + Register(`i`)`

Jej význam je takýto: Spočítaj obsahy registrov `Register(`i`)` a `Register(`j`)` a výsledný súčet ulož do registra `Register(`n`)`. Pri vykonávaní tejto inštrukcie sa pôvodný obsah n-tého registra prepíše výsledkom sčítania. Ostatné registre nemenia svoj obsah. Výnimkou je len `Register(0)`, ktorého obsah sa zvýši o 1, čo znamená, že vykonávanie programu pokračuje v nasledujúcom riadku. Vstupný rad zostáva takisto nezmenený.

Príklad 2.3 Predstavme si situáciu, v ktorej máme v registri `Register(0)` číslo 5 a každý `Register(`i`)` obsahuje číslo i pre $i = 1, 2, 3, 4, 5$ (obr. 2.4a). Všetky ostatné registre obsahujú číslo 0. Piaty riadok programu obsahuje nasledujúcu inštrukciu:

`Register(7)` $\leftarrow$ `Register(1) + Register(4)`

Obrázok 2.4b ukazuje situáciu, ktorá nastane po vykonaní tejto inštrukcie sčítania.



obr. 2.4

Číslo 1 z prvého registra a číslo 4 zo štvrtého registra sa sčítajú (1+4=5)

a výsledok 5 sa uloží do siedmeho registra. Obsahy registrov `Register(1)` a `Register(4)` sa pritom nezmenia. Obsah 0-tého registra sa zvýši z 5 na 6.

Predstavme si situáciu, keď je v šiestom riadku programu nasledujúca inštrukcia:

`Register(7)` ← `Register(1) + Register(7).`

Obsah prvého registra je 1 a obsah siedmeho registra je 5. To znamená, že počítač spočíta $1 + 5 = 6$ a do siedmeho registra uloží číslo 6. Tým sa vymaže pôvodný obsah 5 siedmeho registra. Na tomto príklade vidíme, že výsledok sčítavania smieme uložiť tiež do jedného z tých registrov, v ktorých sa pôvodne nachádzali sčítance. Pritom samozrejme stratíme pôvodnú hodnotu jedného zo sčítancov. □

Úloha 2.4 Predstavme si situáciu po vykonaní prvého sčítania (piateho riadku) v príklade 2.3. Táto situácia je znázornená na obrázku obr. 2.4b. Načrtnite zodpovedajúcu situáciu po vykonaní druhého sčítania v šiestom riadku programu. Potom vykonajte nasledujúce tri operácie:

```
7 Register(3) ← 101
8 Register(3) ← Register(3) + Register(3)
9 Register(3) ← Register(7) + Register(3)
```

Vypíšte stav pamäte, aký bude na konci, po vykonaní všetkých týchto inštrukcií.

Podobne ako súčet môžeme realizovať aj nasledujúce aritmetické operácie:

(4) `Register(`n`)` ← `Register(`j`) - Register(`i`)`

Vykonanie tejto operácie znamená, že sa odčíta obsah i-teho registra od obsahu j-teho registra a výsledok sa uloží do n-tého registra.

(5) `Register(`n`)` ← `Register(`j`)` $*$ `Register(`i`)`

Obsah registrov `Register(`j`)` a `Register(`i`)` sa vynásobí a výsledok sa uloží do n-tého registra.

(6) `Register(`n`)` ← `Register(`j`) / Register(`i`)`

Obsah j-teho registra sa vydelí obsahom i-teho registra a výsledok sa zapamätá v n-tom registri.

(7) `Register(`n`)` ← $\sqrt{\texttt{Register}(n)}$

Vypočíta sa odmocnina čísla uloženého v n-tom registri a výsledok sa uloží do n-tého registra.[3]

Úloha 2.5 Vo všetkých registroch okrem registra `Register(0)` je uložené číslo 0. `Register(0)` obsahuje číslo 1. Vo vstupnom rade stoja dve čísla a a b. Vysvetlite vlastnými slovami, aký výsledok je uložený v treťom registri po vykonaní nasledujúceho programu.

```
1 Načítaj do Register(1)
2 Register(1) ← Register(1) ∗ Register(1)
3 Načítaj do Register(2)
4 Register(2) ← Register(2) ∗ Register(2)
5 Register(3) ← Register(1) + Register(2)
```

Podobne ako v kuchárskych receptoch, ani v počítačových algoritmoch nebude stačiť len vykonávanie istých činností, akými sú v počítači napríklad aritmetické operácie. Potrebujeme tiež možnosť testovať a na základe výsledkov testu určiť ďalší postup. V podstate nám stačia nasledovné dva základné testy:

(8) `Ak Register(`n`) = 0, tak pokračuj v riadku` j

Ak sa obsah n-tého registra rovná 0, tak počítač uloží do 0-tého registra číslo j, a pokračuje vykonávaním inštrukcie v j-tom riadku. Ak je obsah n-tého registra rôzny od nuly, počítač k obsahu 0-tého registra pripočíta jednotku, a pokračuje v nasledujúcom riadku.

(9) `Ak Register(`n`)` $\leq$ `Register(`m`), tak pokračuj v riadku` j

Ak obsah n-tého registra nie je väčší ako obsah m-tého registra, tak do 0-tého registra uložíme číslo j a počítač pokračuje vykonávaním inštrukcie v j-tom riadku. V opačnom prípade program pokračuje vykonávaním inštrukcie v nasledujúcom riadku.

Inštrukcia:

(10) `Pokračuj v riadku` j

je bezpodmienečným príkazom na pokračovanie vykonávania programu v j-tom riadku.

[3] Vypočítanie odmocniny nepatrí medzi bežné základné počítačové operácie. My ju zahrnieme do nášho základného zoznamu operácií, pretože ju potrebujeme pre výpočet riešení kvadratických rovníc. Samozrejme, že počítač je schopný vypočítať odmocninu daného čísla. Tento výpočet sa ale realizuje pomocou programu, ktorý je zostavený len zo základných aritmetických operácií.

Okrem predchádzajúcich inštrukcií potrebujeme ešte príkazy na vypisovanie výsledkov.

(11) `Výstup ← Register(`j`)`

Obsah j-teho registra sa vypíše na výstupné médium.

(12) `Výstup ← „Text“`

Na výstupné médium sa vypíše Text v úvodzovkách. Napríklad príkaz:

`Výstup ← „Ahoj“`

spôsobí, že sa na výstupné médium napíše „Ahoj“ (na výstupnej páske sa objaví text „Ahoj“).

Posledná operácia je:

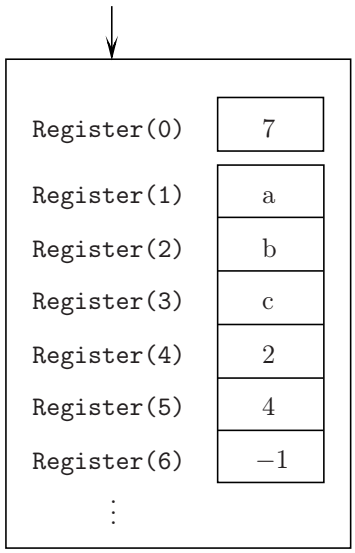
(13) `End`

Táto operácia znamená definitívne ukončenie práce počítača na vykonávaní príkazov daného programu.

Už sme pokročili dosť ďaleko, aby sme mohli naprogramovať algoritmus na riešenie kvadratických rovníc. Informáciu o zmenách obsahu jednotlivých registrov kvôli prehľadnosti uvádzame v kučeravých zátvorkách.

Vstup: Čísla a, b, c
Program:

```
1 Načítaj do Register(1)
  {Register(1) obsahuje a}

2 Načítaj do Register(2)
  {Register(2) obsahuje b}

3 Načítaj do Register(3)
  {Register(3) obsahuje c}

4 Register(4) ← 2

5 Register(5) ← 4

6 Register(6) ← -1
  {Stav pamäti je znázornený na obrázku obr. 2.5}

7 Register(7) ← Register(2) * Register(2)
  {Register(7) teraz obsahuje b^2}
```



obr. 2.5

8 `Register(8)` ← `Register(5) * Register(1)`
{`Register(8)` obsahuje $4a$}

9 `Register(8)` ← `Register(8) * Register(3)`
{`Register(8)` obsahuje $4ac$}

10 `Register(8)` ← `Register(7) - Register(8)`
{`Register(8)` teraz obsahuje $b^2 - 4ac$, a tým je ukončené vykonávanie prvého kroku metódy na riešenie kvadratických rovníc.}

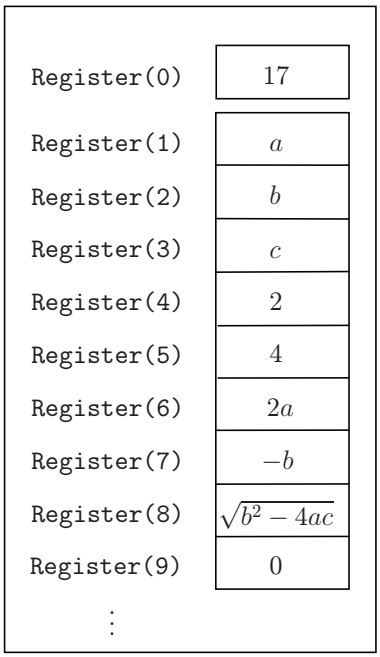
11 `Ak Register(9)` $\leq$ `Register(8), tak pokračuj v riadku 14`
{Register(9) nebol zatiaľ použitý. Pretože doposiaľ nepoužité registre obsahujú vždy číslo 0, v prípade (keď má kvadratická rovnica riešenie) $b^2 - 4ac \geq 0$ sa presúva vykonávanie programu do riadka 14. Keď $b^2 - 4ca < 0$ (keď kvadratická rovnica nemá riešenie), pokračuje výpočet vykonávaním inštrukcie v nasledujúcom dvanástom riadku.}

12 `Výstup` ← „`Neexistuje riešenie.`“

13 `End`
{Po oznámení neexistencie riešenia ukončí program svoju prácu.}

14 `Register(8)` ← $\sqrt{\texttt{Register(8)}}$
{`Register(8)` obsahuje teraz číslo $\sqrt{b^2 - 4ac}$.}

15 `Register(7) ← Register(2) * Register(6)`
{Teraz obsahuje `Register(7)` číslo $-b$. Predchádzajúci obsah b^2 tohto registra sa pritom vymazal.}



obr. 2.6

16 `Register(6) ← Register(1) * Register(4)`
{Situácia je nakreslená na obrázku obr. 2.6.}

17 `Register(11) ← Register(7) + Register(8)`

18 `Register(11) ← Register(11) / Register(6)`
{Teraz obsahuje `Register(11)` prvé riešenie $x_1 = \frac{-b+\sqrt{b^2-4ac}}{2a}$.}

19 `Výstup ← Register(11)`

20 `Register(12) ← Register(7) - Register(8)`

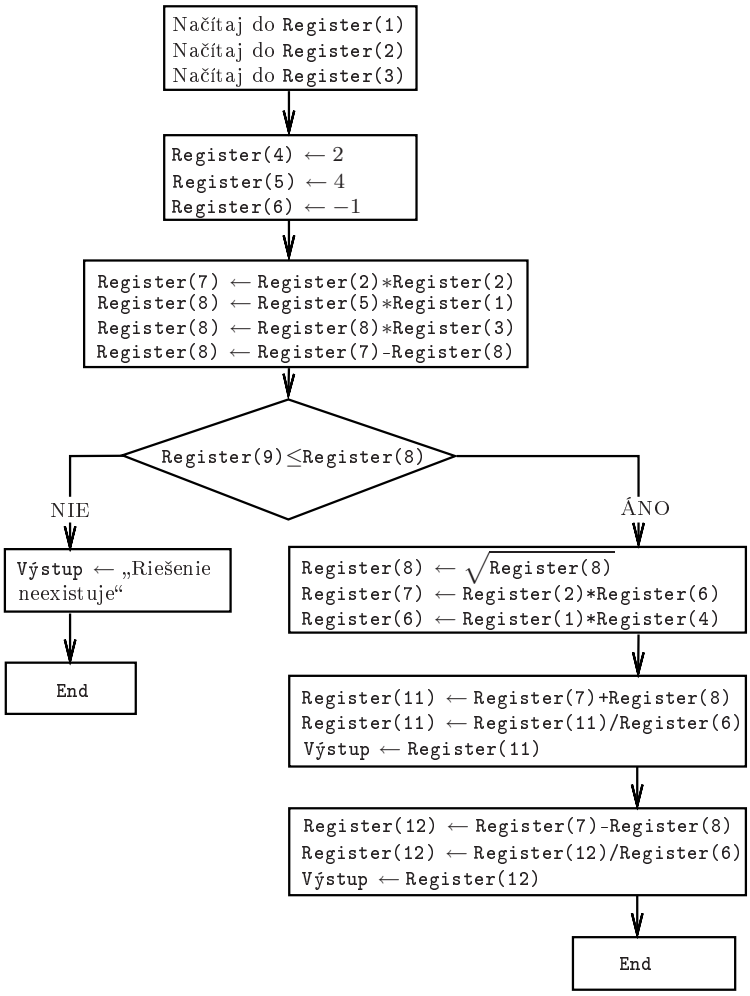
21 `Register(12) ← Register(12) / Register(6)`
{Teraz obsahuje `Register(12)` druhé riešenie $x_2 = \frac{-b-\sqrt{b^2-4ac}}{2a}$.}

22 `Výstup ← Register(12)`

23 `End.`

Grafické znázornenie tohto programu nájdeme na obrázku obr. 2.7.

Úloha 2.6 Určte obsah všetkých registrov po ukončení programu.



obr. 2.7

Úloha 2.7 Ak platí $b^2 - 4ac = 0$, tak existuje len jedno riešenie kvadratickej rovnice a teda $x_1 = x_2$. Upravte program tak, aby v tomto prípade vypísal text „Existuje len jedno riešenie a to“ a príslušné číslo x_1. V prípade $b^2 - 4ac > 0$ chceme, aby program dodatočne vypísal text „Existujú dve riešenia“.

Úloha 2.8 Vysvetlite vlastnými slovami, čo počíta nasledovný program.

```
1 Načítaj do Register(1)
2 Načítaj do Register(2)
```

```
3 Načítaj do Register(3)
4 Register(4) ← Register(1) + Register(2)
5 Register(4) ← Register(3) + Register(4)
6 Register(5) ← 3
7 Register(6) ← Register(4) / Register(5)
8 Výstup ← Register(6)
9 End
```

Určte a v tabuľke znázornite vývoj obsahov jednotlivých registrov po každom kroku programu.

Úloha 2.9 Napíšte program, ktorý pre dané číslo x vypočíta nasledovnú hodnotu:

$$3x^2 - 7x + 11.$$

Úloha 2.10 Napíšte program, ktorý pre dané 4 čísla a, b, c, x vypočíta hodnotu

$$ax^2 + bx + c.$$

Úloha 2.11 Napíšte program, ktorý pre dané 4 čísla a, b, c, d vo vstupnom rade určí ich maximum (najväčšiu z týchto 4 hodnôt).

Nie je nevyhnutnosťou zapísať každý algoritmus vo forme programu len kvôli tomu, aby sme sa presvedčili, že ide skutočne o algoritmus. Napríklad keď je na prvý pohľad zrejmé, že metóda na riešenie kvadratických rovníc sa dá realizovať pomocou aritmetických operácií a testov na porovnávanie čísiel, a že táto metóda rieši každý prípad tejto úlohy, tak ju môžeme pokladať za algoritmus. *Napísanie zodpovedajúceho programu (naprogramovanie algoritmu) považujeme za transformáciu zápisu algoritmu do reči počítača.* Z formálneho matematického hľadiska ale možno považovať túto transformáciu prezentácie algoritmu za dôkaz strojovej (automatickej) realizovateľnosti daného algoritmu.

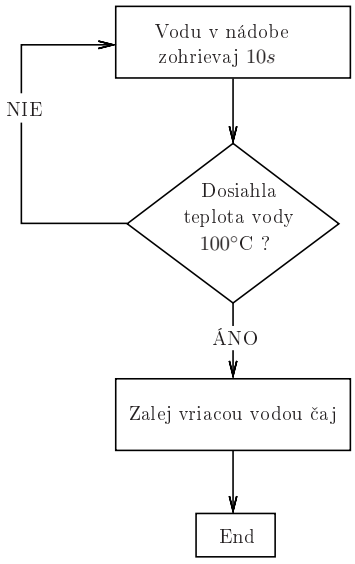
2.4 Ako neúmyselne zakliať program na večné bezvýsledné počítanie?

Jednou z najdôležitejších požiadaviek na definíciu algoritmu na riešenie nejakého problému (úlohy) je, aby algoritmus vždy svoju prácu ukončil v konečnom čase a na výstup dal výsledok. V odbornej informatickej terminológii hovoríme o **zastavení**. Ak algoritmus A pracuje na vstupe x konečne dlho a potom svoju prácu na x definitívne ukončí, tak hovoríme, že **algoritmus A zastaví na prípade problému x.** Ak algoritmus

A zastaví na každom možnom vstupe, tak hovoríme, že algoritmus ***A*** **zastaví vždy**. Vyjadrené slovami informatiky, od každého algoritmu vyžadujeme, aby vždy zastavil.

Prirodzene môžete s počudovaním namietať: „To je predsa samozrejmé. Kto by už len vyvíjal programy, ktoré pracujú nekonečne dlho, dôsledkom čoho nikdy nedajú nijaký výstup?“ Potrebujeme sa tým vôbec zaoberať? Problém je ale v tom, že nie je veľké umenie vyvinúť neúmyselne program, ktorý pre nejaké špeciálne, ale možné vstupy začne do nekonečna opakovať nejaký cyklus. Ako sa môže niečo také stať profesionálnemu programátorovi? Veľmi ľahko. Napríklad stačí, ak zabudne zobrať do úvahy nejaké špeciálne situácie, ktoré môžu, síce zriedkavo, ale predsa len za istých okolností, nastať. Aby sme videli, ako ľahko sa nám čosi také môže pritrafiť, vráťme sa k našim kuchárskym algoritmom.

Naším cieľom je, aby voda zovrela, a potom ju použiť na prípravu čaju. Pritom chceme zaobchádzať opatrne s energiou a nenechať vodu vrieť zbytočne dlhšie ako 20 sekúnd. Na tento účel môžeme navrhnúť program znázornený na obrázku obr. 2.8.



obr. 2.8

Na prvý pohľad sa nám zdá, že je všetko v poriadku a algoritmus pokladáme za funkčný. Ale len dovtedy, pokiaľ sa nerozhodne tento recept na varenie čaju použiť nejaký horolezec na vrchole Matterhornu. V tejto nadmorskej výške je nižší tlak, a teda voda vrie pri nižšej teplote a pri da-

nom tlaku nemôže dosiahnuť teplotu 100°C. Takže test nášho programu nikdy nebude splnený. Samozrejme, že horolezec nebude v pravom slova zmysle variť čaj donekonečna, lebo buď sa mu minie palivo, alebo sa vyparí voda z hrnca.

Všetci vidíme, kde sa stala chyba. Jednoducho sme pri písaní receptu nemysleli na túto špeciálnu situáciu. A presne to isté sa môže stať každému programátorovi, ak nemyslí neustále na všetky prípady danej úlohy a na všetky možné špeciálne situácie, ktoré môžu v priebehu výpočtov nastať. Z podobných dôvodov spadla už aj jedna raketa, lebo programátori nepočítali s tým, že pri výpočtoch ich programov môžu vzniknúť také veľké čísla, ktoré sa nezmestia do 32-bitového registra. Na ukážku ale uvedieme jednoduchší príklad.

Príklad 2.4 Na začiatku obsahuje `Register(0)` číslo 1 a všetky ostatné registre obsahujú číslo 0. Vo vstupnom rade čakajú čísla a a b. Naprogramovali sme nasledujúci program.

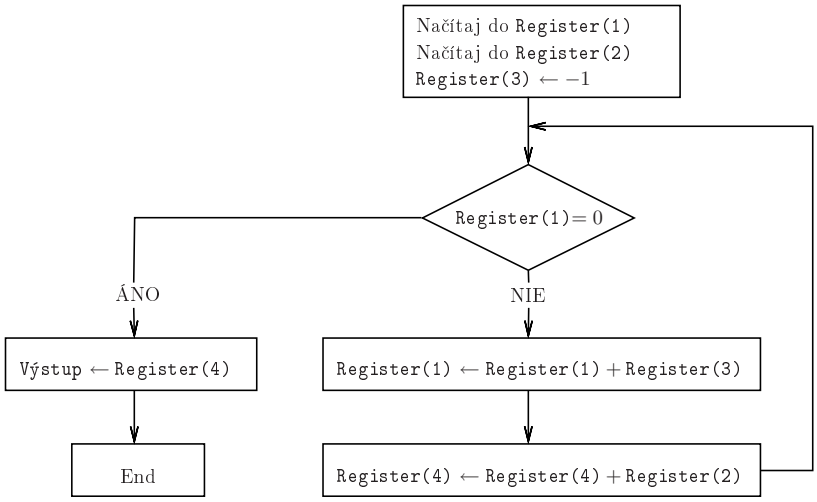
```
1 Načítaj do Register(1)
2 Načítaj do Register(2)
3 Register(3) ← -1
4 Ak Register(1) = 0, tak pokračuj v riadku 8
5 Register(1) ← Register(1) + Register(3)
6 Register(4) ← Register(4) + Register(2)
7 Pokračuj v riadku 4
8 Výstup ← Register(4)
9 End
```

Zodpovedajúce grafické znázornenie programu nachádzame na obrázku obr. 2.9.

Úlohou algoritmu je vypočítať súčin $a \cdot b$ len pomocou sčítania a odčítania. V podstate náš program počíta

$$\underbrace{b + b + b + \ldots + b}_{a \text{ krát}},$$

t.j. spočítava a-krát hodnotu b.



obr. 2.9

Úloha 2.12 Uvažujme o konkrétnych vstupoch $a = 3$ a $b = 7$. Vykonajte výpočet programu na tomto vstupe. Zostavte tabuľku, ktorá ukazuje po každom kroku výpočtu (po vykonaní každej inštrukcie) stavy všetkých registrov.

Ak $a = 0$, tak výsledok má byť $a \cdot b = 0 \cdot b = 0$. To sa aj naozaj stane, lebo a sa načíta do prvého registra a test vo štvrtom riadku potom vedie priamo do ôsmeho riadku, v ktorom program dá na výstup obsah štvrtého registra, čo je v tomto prípade číslo 0.
Ak $a > 0$, pripočítame v šiestom riadku programu číslo b k obsahu štvrtého registra, v ktorom má na konci ostať definitívny výsledok. V piatom riadku programu pritom zmenšíme obsah prvého registra o hodnotu 1. V prvom registri bolo pôvodne uložené číslo a, a tak po i opakovaniach cyklu pre $i < a$ je v prvom registri číslo $a - i$ a vo štvrtom registri je uložené číslo

$$\underbrace{b + b + \ldots + b}_{i \text{ krát}} = i \cdot b.$$

Ak „`Register(1)` $= 0$“, tak po opustení cyklu vieme, že cyklus sme realizovali práve a-krát a teda `Register(4)` obsahuje hľadanú hodnotu:

$$\underbrace{b + b + b + \ldots + b}_{a \text{ krát}} = a \cdot b.$$

Takže vidíme, že sa nám podarilo vyvinúť program, ktorý dokáže násobiť dve čísla bez použitia operácie násobenia zo zoznamu našich základných

operácií. To znamená, že keď z nášho zoznamu operácií vynecháme operáciu násobenia, neoslabíme schopnosti počítača, a teda ani nášho pojmu algoritmus.
Program na obrázku obr. 2.9 má ale predsa jeden zádrheľ. Na začiatku sme povedali, že a a b sú celé čísla. Čo sa stane, ak je jedno z čísiel, alebo obe čísla a, b záporné? Ak je b záporné a a je kladné, cyklus sa zopakuje presne a-krát a dostaneme správny výsledok. Ak je ale a záporné, bude program donekonečna vykonávať cyklus. Prečo? `Register(1)` bude obsahovať na začiatku záporné číslo. V priebehu cyklu sa toto číslo bude vždy o jednotku zmenšovať, a teda obsah prvého registra sa nemôže nikdy zväčšiť na hodnotu 0. □

Úloha 2.13 Ako možno upraviť program z obrázka 2.9 tak, aby korektne vynásobil dve čísla a a b pomocou sčítania a odčítania, aj v prípade, keď je číslo a záporné?

Úloha 2.14 Pokúste sa napísať program, ktorý vypočíta pre dve kladné celé čísla a a b súčet $a + b$, len pomocou nasledujúcich operácií

`Register(`i`)` $\leftarrow$ `Register(`i`)+1`
`Register(`j`)` $\leftarrow$ `Register(`j`)-1,`

ktoré zväčšujú alebo zmenšujú obsah registrov o jedna. Okrem týchto dvoch inštrukcií nie sú v programe dovolené nijaké iné aritmetické operácie.

Nakoniec vidíme, že všetky algoritmy sa dajú prepísať do programu, ktorý používa len testovanie obsahu registra na 0, pripočítanie a odpočítanie jednotky a vstupno-výstupné operácie. Z tohto uhla pohľadu nemožno mať nijaké pochybnosti o tom, že to, čo označujeme za algoritmy, možno naozaj automaticky realizovať na počítači.

Zvyšok tejto kapitoly je venovaný len tým, ktorí by radi presnejšie vedeli, ako vyzerá skutočný repertoár príkazov počítača. Na začiatok upozorníme na to, že z nášho doterajšieho zoznamu treba vynechať operáciu počítania odmocniny. Na vypočítanie odmocniny nejakého čísla je nevyhnutné napísať program, ktorý ju vypočíta len pomocou základných aritmetických operácií $+, -, *$ a $/$. Takýto program nebudeme písať, pretože to vyžaduje viac práce, ako by sa na prvý pohľad mohlo zdať.

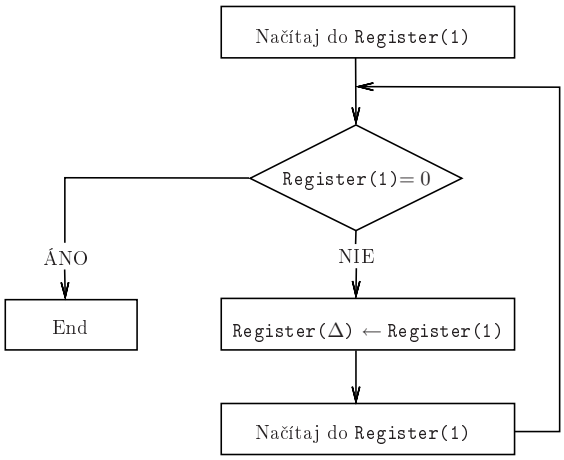
Na druhej strane nám ale chýba ešte zopár inštrukcií, ktoré na realizáciu istých úloh dokonca nevyhnutne potrebujeme. Pozrime sa na nasledujúcu úlohu. Ako vstup dostaneme postupnosť celých čísiel. Vopred ale nevieme, koľko čísiel sa nachádza v tejto postupnosti. Koniec postupnosti rozpoznáme len vďaka dohode, že všetky čísla v postupnosti sú rôzne

od nuly a samotné číslo 0 označuje koniec postupnosti. Našou úlohou je všetky čísla tejto postupnosti načítať a postupne uložiť do registrov `Register(100)`, `Register(101)`, `Register(102)`, atď. To znamená, že i-te číslo postupnosti máme uložiť do registra s adresou $100 + i - 1$. Tento proces máme ukončiť, ak načítame číslo 0. Pri písaní zodpovedajúceho programu by sme mohli vyskúšať začať nasledovným spôsobom.

```
1 Načítaj do Register(1)
2 Ak Register(1) = 0, tak pokračuj v riadku □
3 Register(100) ← Register(1)
4 Načítaj do Register(1)
5 Ak Register(1) = 0, tak pokračuj v riadku □
6 Register(101) ← Register(1)
7 Načítaj do Register(1)
8 Ak Register(1) = 0, tak pokračuj v riadku □
9 Register(102) ← Register(1)
  ⋮
```

Naša stratégia bude takáto: Nasledujúce číslo zo vstupného radu vždy prečítame do `Register(1)` a keď je toto číslo rôzne od nuly, tak ho uložíme do najbližšieho voľného registra od adresy 100 smerom nahor. Jedinou otázkou je, ako pokračovať ďalej. Keď je vstupná postupnosť $17, -6, 0$, musíme prácu ukončiť. Ak má ale vstupná postupnosť 10000 čísiel, mal by mať náš program 30000 riadkov. Nevieme ale, kedy máme ukončiť písanie programu, a teda ani nevieme, do ktorého riadka máme napísať inštrukciu `END`. Preto sme použili v programe označenie □, lebo sme jednoducho nevedeli, do ktorého riadka môžeme `END` uložiť. Jedno je ale jasné. Nemôžeme napísať nekonečný program. Východiskom by mohlo byť použitie cyklu znázorneného na obrázku 2.10.

Problém je len v tom, že nevieme, v ktorom registri `Register(`△`)` máme uložiť práve načítané číslo. Adresa △ nemôže byť stále tá istá, pretože si chceme zapamätať všetky čísla postupnosti, teda každé číslo na inú adresu. V podstate máme v i-tom prechode cyklom uložiť naposledy prečítané číslo do registra s adresou $100 + i - 1$. To ale nemôžeme zrealizovať, pretože naše inštrukcie nám umožňujú nahradiť symbol △ len jedným konkrétnym číslom, ktoré sa nemení.



obr. 2.10

Z týchto dôvodov majú počítače v základnom vybavení inštrukcie takzvanej **nepriamej adresácie**. Vykonanie inštrukcie

(14) `Register(Register(`i`)) ← Register(`j`)`

pre adresy i a j spôsobí, že sa obsah j-tého registra uloží do registra, ktorého adresa je obsahom i-teho registra.

Je to komplikované? Pozrime sa na to na konkrétnom príklade. Nech obsah registra `Register(3)` je 112 a obsah siedmeho registra je 24. Počítač má vykonať príkaz.

`Register(Register(3)) ← Register(7).`

Najprv si počítač pozrie obsah tretieho registra a zistí, že tento obsah je 112. Potom už len vykoná nám známy príkaz.

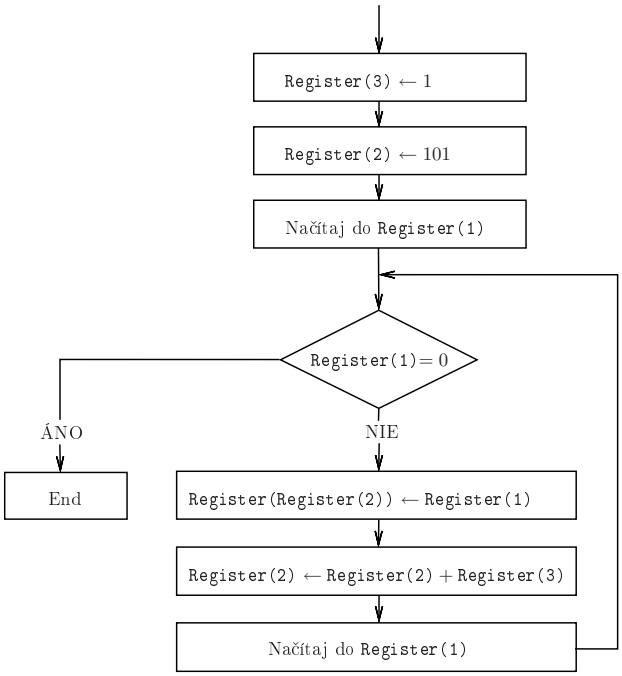
`Register(112) ← Register(7).`

Takže číslo 24, ktoré je obsahom siedmeho registra sa zapamätá v registri `Register(112)`. Okrem nového obsahu 24 v 112-tom registri sa nemení obsah nijakého registra okrem nultého, ktorého hodnota sa ako zvyčajne zvýši o 1.

Úloha 2.15 Pre väčšinu doteraz predstavených počítačových inštrukcií existuje variant s nepriamou adresáciou. Pokúste sa sami vysvetliť význam nasledujúcich príkazov:

a) `Register(`k`) ← Register(Register(`m`))`
b) `Register(Register(`i`)) ← Register(`l`)*Register(`j`).`

Pomocou nepriamej adresácie sa dá vyriešiť aj náš problém, keď si chceme zapamätať neznámy počet údajov. Riešením je program na obrázku 2.11. V druhom registri máme stále uloženú aktuálnu adresu, na ktorú chceme uložiť nasledujúce číslo zo vstupného radu. Na začiatku do druhého registra uložíme adresu 100 a po každom zapamätaní ďalšieho čísla zvýšime o jednotku obsah druhého registra. Číslo 1 je po celý čas uložené v treťom registri.



obr. 2.11

Úloha 2.16 Vo vstupnom rade čakajú čísla 113,-2,20,8,0. Simulujte krok po kroku program na obrázku 2.11 a načrtnite pritom aktuálne obsahy registrov s adresami 0, 1, 2, 3, 100, 101, 102, 103, 104 a 105. Predpokladáme, že na začiatku všetky registre obsahujú číslo 0.

2.5 Zhrnutie alebo čo sme sa tu naučili

Či mi to veríte, alebo nie, ak ste úspešne vyriešili sformulované úlohy, tak ste už aj trochu programovali, a teda ste si aj vybudovali akú-takú predstavu o tom, čo to znamená pracovať ako programátor. To ale nebolo hlavným cieľom tejto kapitoly.

Naším cieľom bolo ujasniť si význam pojmu algoritmus v zmysle formalizácie pojmu metódy. Pochopili sme, že naše očakávania na definíciu tohto pojmu zodpovedajú nasledujúcim požiadavkám:

1. Metóda ako algoritmus na riešenie nejakého problému sa musí dať úspešne aplikovať aj keď jej užívateľ nie je expertom na riešenie daného problému. Jednoducho netreba rozumieť, prečo algoritmus dosiahne stanovený cieľ. Stačí, ak vieme vykonávať jednoduché činnosti, z ktorých je algoritmus zostavený. V definícii algoritmu musia byť uvedené v zozname inštrukcií a musí platiť všeobecný konsenzus o tom, že každá z týchto inštrukcií (činností) je taká jednoduchá, že sa dá vykonávať automaticky (strojom).

2. Algoritmus nenavrhujeme s cieľom vyriešiť nejaký špecifický prípad problému. Algoritmus na riešenie daného problému musí byť schopný úspešne riešiť všetky možné prípady problému, ktorých je bežne nekonečne veľa. (Na zopakovanie: Problém je všeobecne postavená úloha ako triedenie čísiel, alebo riešenie kvadratických rovníc. Prípad problému je konkrétna úloha ako „Usporiadaj vzostupne čísla $1, 7, 3, 2, 8$“ alebo „Nájdi riešenie kvadratickej rovnice $2x^2 - 3x + 5 = 0$“.)

3. Pre algoritmus na riešenie daného problému musíme mať istotu, že tento algoritmus opisuje úspešnú cestu riešenia každého prípadu problému. To znamená, že algoritmus ukončí prácu na každom vstupe v konečnom čase a jeho výstup vždy zodpovedá správnemu výsledku.

Každý algoritmus sa dá zapísať ako program v nejakom programovacom jazyku. Program je takým opisom algoritmu, ktorý je zrozumiteľný pre počítač. Pritom pojem program nie je synonymom pojmu algoritmus. Program nie je nič iné, ako postupnosť inštrukcií. Táto postupnosť inštrukcií nemusí dávať zmysel. Napríklad vykonávanie nejakého programu môže viesť k nezmyselným výpočtom, ktoré neriešia nijaký problém, alebo k nekonečnému opakovaniu jednej a tej istej činnosti.

Vzorové riešenia k vybraným úlohám

Úloha 2.2 Kuchársky algoritmus na zohriatie 1 litra vody na aspoň 90°C môžeme opísať takto:

```
1 Nalej 1 liter vody do hrnca H.
```

```
2 Postav hrniec H na 15 sekúnd na horúcu platničku
  a potom ho z platničky zodvihni.
3 Ak teplota vody dosiahla aspoň 90°C, ukonči prácu.
  Ak nie, pokračuj v práci činnosťou 2.
```

Úloha 2.3 Stav pamäti po vykonávaní jednotlivých inštrukcií názorne priblížime pomocou nasledujúcej tabuľky:

	1	2	3	4	5
Vstup	11, 12, 13	12, 13	12, 13	13	
Register(0)	1	2	3	4	5
Register(1)	0	11	11	11	11
Register(2)	1117	1117	100	100	13
Register(3)	21	21	21	12	12
Register(4)	0	0	0	0	0

Prvý stĺpec tabuľky zodpovedá situácii pred štartom programu. Stĺpec $i+1$ opisuje stav pamäti a vstupného radu tesne po vykonaní i-tej inštrukcie programu, teda pred vykonávaním $(i+1)$-tej inštrukcie.

Úloha 2.8 Daný program najprv načíta tri hodnoty zo vstupného radu (inštrukcie v riadkoch 1, 2, a 3). Potom vypočíta ich súčet a uloží ho do štvrtého registra (riadky 4 a 5). V riadkoch 6 a 7 sa vypočíta priemer načítaných hodnôt a výsledok sa uloží do šiesteho registra. Príkaz v ôsmom riadku vypíše vypočítaný priemer na výstup. Nasledujúca tabuľka, podobne ako tabuľka v úlohe 2.3, ukazuje vývoj situácie po vykonaní jednotlivých inštrukcií. Aby sme zvýšili prehľadnosť, uvádzame hodnoty registrov len vtedy, ak sa zmenil v tomto kroku výpočtu ich obsah.

Vstup	a, b, c	b, c	c							
Register(0)	1	2	3	4	5	6	7	8	9	10
Register(1)	0	a								
Register(2)	0		b							
Register(3)	0			c						
Register(4)	0				$a+b$	$a+b+c$				
Register(5)	0						3			
Register(6)	0							$\frac{a+b+c}{3}$		
Výstup									$\frac{a+b+c}{3}$	

Veľa by sa toho vo svete neudialo, keby sme sa neustále iba strachovali aké dôsledky bude mať naše úsilie.

Georg Christoph Lichtenberg

Kapitola 3

Nie je nekonečno ako nekonečno, alebo prečo je nekonečno v informatike nekonečne dôležité?

3.1 Prečo potrebujeme nekonečno?

Pre nás známy vesmír je konečný a väčšina fyzikálnych teórií je založená na predstave konečného vesmíru. Všetko to, čo vidíme, čoho sa dotýkame a s čím prichádzame do kontaktu je konečné.

> *Na čo je potom dobré nekonečno?*
> *Je nekonečno niečo neprirodzené a vykonštruované, jednoducho hračka matematiky?*

Napriek možným pochybnostiam pri prvom stretnutí s konceptom nekonečna, dovolíme si tvrdiť, že nekonečno je úspešný nástroj na skúmanie reálneho konečného sveta. Naše prvé stretnutie s nekonečnom absolvujeme väčšinou už na prvom stupni základnej školy, kde sa zoznamujeme s množinou všetkých prirodzených[1] čísiel:

$$\mathbb{N} = \{0, 1, 2, 3, \ldots\}.$$

Základný princíp definovania tejto množiny je jednoduchý:

[1] V tejto knihe budeme považovať číslo 0 za prirodzené číslo.

Pre každé prirodzené číslo i existuje o jednotku väčšie prirodzené číslo $i + 1$.

Vyjadrené inými slovami, neexistuje najväčšie číslo (ktoré by bolo väčšie ako všetky ostatné), pretože pre každé číslo existujú od neho väčšie čísla. Čo to znamená? Jednoducho nie je možné zapísať všetky prirodzené čísla jedno za druhým, pretože nezávisle od toho, koľko čísiel sme už zapísali, stále nasledujú ďalšie a ďalšie. Tak môžeme pokračovať bez toho, aby sme skončili, a preto hovoríme o **potenciálnom nekonečne** alebo o **neobmedzenom** počte prirodzených čísel. Podobne je to s priamkou v geometrii. Priamka je potenciálne nekonečná, a teda má nekonečnú dĺžku, pretože môžeme po nej pochodovať neobmedzene dlho. Na jej koniec nikdy neprídeme a je jedno, kde sa na nej nachádzame, vždy môžeme v chôdzi pokračovať ďalej v tom istom smere.

Hlavný problém s konceptom nekonečna je v našej neschopnosti si nekonečno predstaviť. **Aktuálne nekonečno** nemôžeme jednoducho vidieť naraz celé. Chápeme, že máme nekonečne veľa (neobmedzene veľa) prirodzených čísiel, ale nie sme schopní vidieť naraz všetky prirodzené čísla. Presne tak isto, ako nemôžeme naraz vidieť celú nekonečnú priamku. Sme schopní pozorovať vždy len nejaký konečný zlomok (konečnú časť) uvažovaného nekonečného objektu. Napriek tomu označujme nekonečné objekty pomocou symbolov a potom pracujeme s týmito symbolmi ako s konečnými reprezentáciami nekonečných objektov.

Na tomto mieste by niekto mohol navrhnúť nahradiť koncept nekonečna pomocou jedného obrovského, ale konečného ohraničenia. Napríklad, môžeme sa rozhodnúť definovať ako najväčšie prirodzené číslo počet[2] protónov v známom vesmíre. Takto ohraničené množstvo čísiel nám vo väčšine počtových úkonov a úvah bude aj stačiť. Ale v okamihu, keď sa budeme snažiť vypočítať energiu celého vesmíru, alebo sa budeme chcieť zaoberať všetkými možnými vzťahmi medzi elementárnymi časticami, nám takto ohraničené čísla nebudú stačiť. Je jedno, aké veľké číslo si zvolíme ako možné ohraničenie počtu čísiel, vždy sa nájdu zmysluplné situácie, ktorých štúdium si vyžaduje prácu s ešte väčšími číslami. Navyše nejde len o to, že si vieme ku každému číslu predstaviť od neho ešte väčšie číslo. Toto väčšie číslo môžeme dokonca vždy aj zapísať, a teda nepochybujeme o jeho existencii. Prečo by sme si mali potom zakazovať niečo, čo existuje a prípadne to môžeme dokonca aj potrebovať?

Ak ale chceme čitateľa presvedčiť o užitočnosti konceptu nekonečna, potrebujeme predložiť viac argumentov ako len prirodzenú existenciu po-

[2]Toto číslo pozostáva zo 79 decimálnych číslic.

tenciálneho nekonečna. Dovolíme si tvrdiť, že vďaka konceptu nekonečna dokážeme náš konečný svet úspešnejšie skúmať a lepšie chápať. Nekonečno nám neumožňuje uvažovať len o nekonečných veľkostiach. Môžeme uvažovať aj o nekonečne malých objektoch.

> *Ktoré číslo je najmenšie kladné racionálne číslo? Inými slovami: „Ktorý zlomok je najmenší zlomok väčší od nuly?“*

Začnime s príkladom zlomku 1/1000. Ten môžeme deliť dvomi a dostaneme zlomok 1/2000, ktorý je menší ako 1/1000. Zlomok 1/2000 môžeme znova vydeliť dvomi a dostaneme zlomok 1/4000. Je jedno nakoľko malý kladný zlomok

$$\frac{1}{x}$$

vezmeme, vždy ho môžeme vydeliť dvomi a dostaneme kladný zlomok,

$$\frac{1}{2x},$$

ktorý je ešte menší ako $1/x$ a pritom stále väčší ako 0. Vidíme, že tento príbeh je bez konca. Ku každému kladnému racionálnemu číslu existuje menšie kladné racionálne číslo, atď.

David Hilbert (1862–1943), jeden z najslávnejších matematikov svojej doby, povedal:

> *„V istom zmysle nie je matematická analýza ničím iným, ako symfóniou na tému nekonečna.“*

A my k tomu pridávame, že bez pojmu nekonečna by nemohla existovať ani fyzika tak, ako ju poznáme. Kľúčové koncepty matematiky ako limita, derivácia, integrál, spojitosť a diferenciálne rovnice sú vybudované na pojme nekonečna a nemôžu bez neho existovať. A bez týchto konceptov nie je fyzika schopná modelovať náš svet. Problémy by nastali už pri definícii základných fyzikálnych pojmov. Ako by sme bez týchto matematických pojmov a konceptov definovali napríklad pojem zrýchlenia? Mnohé z týchto pojmov vznikli práve preto, že ich fyzika potrebovala pre vlastný vývoj.

Dôsledkom našich úvah je, že bez pojmu nekonečna by zmizli veľké časti matematiky. Vzhľadom na to, že matematika je formálny jazyk celej vedy a my dnes spájame istý stupeň „zrelosti“ vedných disciplín so stupňom používania tohto jazyka, vrhlo by vymazanie pojmu nekonečna celú vedu stáročia dozadu.

Rovnako to platí aj pre informatiku. Potrebujeme rozlišovať programy (ktoré pripúšťajú nekonečné výpočty) od algoritmov (ktoré garantujú

ukončenie výpočtov v konečnom čase). Naviac existuje nekonečne veľa programov a nekonečne veľa rôznych výpočtových úloh. Typické výpočtové problémy pozostávajú z nekonečného počtu prípadov problému. Nekonečno je v informatike neodmysliteľné.

Cieľom tejto kapitoly nie je len ukázať, že koncept nekonečna je úspešným nástrojom výskumu v informatike. Ako keby nám nestačilo, že sa musíme trápiť s pochopením potenciálneho a aktuálneho nekonečna, ktoré sme nikdy nevideli a ani neuvidíme, zaťažíme čitateľa ešte náročnejšou otázkou:

> *„Existuje len jedno nekonečno, alebo existuje viac rôzne veľkých nekonečien?“*

Nepreháňame to trochu a nehráme sa na vedcov, ktorí už od dobroty nevedia, čo by mohli robiť a venujú sa nezmyselným umelým konštrukciám? Nie. Táto vysoko abstraktná otázka má pre vedu neuveriteľne veľkú hodnotu. Naším cieľom je predstaviť jeden z najdôležitejších objavov o nekonečne a ukázať, že existujú prinajmenšom dve[3] rôzne veľké nekonečná. Čo tým získame? Vďaka tomuto objavu môžeme ukázať, že počet rôznych algoritmických úloh je väčší ako počet všetkých programov. Takto dostaneme prvý základný výsledok informatiky, ktorý má filozofickú hĺbku a je kľúčovým príspevkom pre celú vedu.

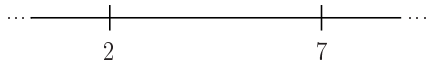
> *Nie všetko je automatizovateľné, pretože existujú úlohy, ktoré sa nedajú riešiť pomocou nijakých algoritmov.*

Vďaka tomuto prvému existenčnému kroku nájdeme v nasledujúcej kapitole konkrétne zmysluplné problémy z praxe, ktoré sa nedajú riešiť algoritmicky, teda automaticky s pomocou výpočtovej techniky. Je to nádherný príklad toho, ako koncept nejakého v realite neexistujúceho objektu môže viesť k objavu, ktorý má nemalý vplyv na prax. Možno ste prekvapení, ale nezabúdajme na nasledujúce. Cesta k vedeckým poznatkom cez hypotetické koncepty vo fyzikálnej realite neexistujúcich abstraktných objektov je skôr typická ako výnimočná. A najdôležitejšie, čo sa počíta, je to, či sme dosiahli náš výskumný cieľ. To zodpovedá presne tomu, čo sme sa v prvej kapitole naučili od Gödela. Potrebujeme nové pojmy, aby sme mohli skúmať a dokazovať pravdivosť vedeckých tvrdení, ktoré s týmito pojmami vo svojej formulácii nemajú nič spoločné.

[3] Existuje nekonečne veľa rôzne veľkých nekonečien.

3.2 Cantorova metóda na porovnávanie veľkosti nekonečien

Porovnávanie (konečných) čísiel je veľmi jednoduché. Všetky čísla ležia na reálnej osi a z dvoch čísiel je vždy menšie to, ktoré sa nachádza vľavo od toho druhého (obr. 3.1).



obr. 3.1

Takže 2 je menšie ako 7, pretože číslo 2 sa nachádza vľavo od čísla 7. To ale nie je koncepcia na porovnávanie čísiel, pretože a priori ukladáme čísla na hypotetickú reálnu os tak, aby ich hodnoty rástli v smere zľava doprava a klesali v smere sprava doľava. Ale na osi ležia len konečne veľké čísla. Je jedno, o ktorom mieste (bode) reálnej osi uvažujeme, vždy na ňom leží jedno konkrétne konečne veľké číslo. A to platí napriek nekonečnosti reálnej osi v oboch smeroch. Toto je koncepcia potenciálneho nekonečna. Na tejto osi môžeme ísť ľubovoľne ďaleko doprava, alebo doľava a jedno na akom mieste sa zastavíme, vždy tam nájdeme jedno konkrétne konečne veľké číslo. Nekonečné čísla na osi nenájdeme. V matematike používame pre označenie nekonečna symbol

$$\infty$$

„ležatá osmička“, ktorý je odvodený od hebrejského symbolu aleph (ℵ). Ak ale veľkosť všetkých nekonečien budeme označovať jedným symbolom ∞, nebudeme mať možnosť veľkosti rôznych nekonečien porovnávať.

Čo robiť?

V tejto situácii potrebujeme novú reprezentáciu čísiel a na to potrebujeme pojem množiny. Množina je kolekcia (zbierka, súbor) rôznych objektov, ktoré nazývame prvkami množiny. Napríklad $\{2, 3, 7\}$ je množina, ktorá obsahuje tri čísla $2, 3$ a 7. Pri označovaní množín používame zložené zátvorky { a }. Množina {Janko, Anička, Peter, Paula} obsahuje štyri objekty (prvky) Janko, Anička, Peter a Paula. Pre každú množinu A označujeme pomocou symbolu

$$|A|$$

počet prvkov v A a hovoríme o **mohutnosti** (**kardinalite**) **množiny** A. Napríklad

$$|\{2, 3, 7\}| = 3 \quad \text{a} \quad |\{\text{Janko, Anička, Peter, Paula}\}| = 4.$$

Čísla budeme reprezentovať pomocou mohutností množín. Takže mohutnosť množiny $\{2, 3, 7\}$ reprezentuje číslo 3. Očividne takýmto spôsobom získavame pre každé kladné celé číslo nekonečne veľa reprezentácií. Napríklad

$$|\{1, 2\}| \ , \ |\{7, 11\}| \ , \ |\{\text{Peter, Paula}\}| \ , \ |\{\square, \bigcirc\}|$$

sú všetko reprezentácie čísla 2. Nie je to trocha prehnané a zbytočne komplikované? Takto pôvodne reprezentovali čísla naši prapredkovia. Čo sme získali touto reprezentáciou, okrem návratu do dávnej minulosti?

Pre porovnávanie konečne veľkých čísiel je táto reprezentácia čísiel možno skutočne archaická a nepraktická. Ale výhodou tejto reprezentácie je, že môžeme porovnávať nekonečné veľkosti. Číslo pre veľkosť $\mathbb{N} = \{0, 1, 2, \ldots\}$ je nekonečne veľké číslo, ktoré reprezentuje počet všetkých prirodzených čísel. Keď pomocou $\mathbb{Q}^+$ označíme množinu všetkých kladných racionálnych čísiel, potom je číslo

$$\left|\mathbb{Q}^+\right|$$

nekonečne veľké číslo, ktoré reprezentuje počet všetkých kladných racionálnych čísiel (kladných zlomkov). Podobne označuje

$$|\mathbb{R}|$$

počet reálnych čísiel, keď predpokladáme, že $\mathbb{R}$ reprezentuje množinu reálnych čísiel. Teraz chápeme prednosti uvedenej reprezentácie čísiel. Odrazu máme možnosť pýtať sa:

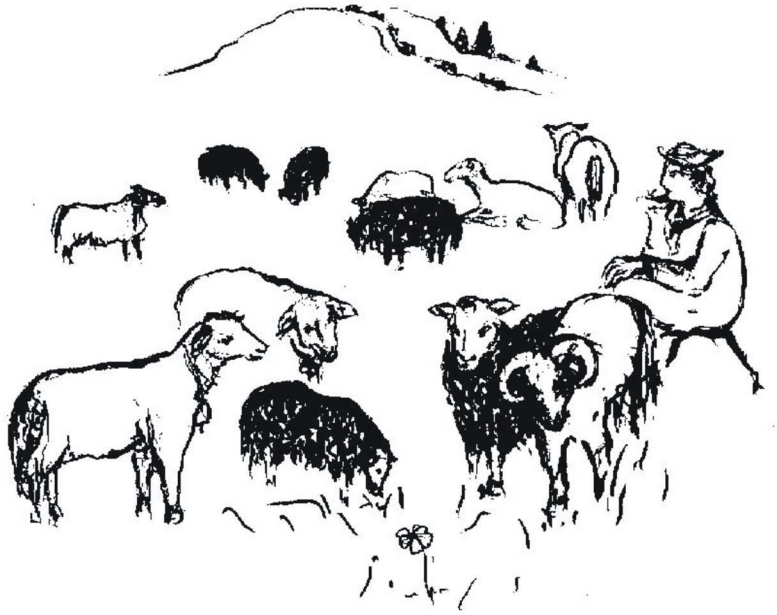
„Je $|\mathbb{N}|$ menšie ako $|\mathbb{R}|$?“

alebo

„Je $\left|\mathbb{Q}^+\right|$ rovnako veľké ako $|\mathbb{R}|$?“

Vďaka tejto reprezentácii čísiel môžeme po prvýkrát sformulovať otázku, či je nejaké nekonečno väčšie ako iné nekonečno.

V tomto okamihu sa nám podarilo zredukovať náš problém porovnávania čísiel na porovnávanie veľkosti (mohutnosti) množín. Ako ale porovnať mohutnosť dvoch množín? Ak sú obe množiny konečné, je to jednoduché. Spočítame počet prvkov v oboch množinách a porovnáme bežným spôsobom zodpovedajúce čísla. V prípade nekonečných množín tento postup ale nevedie k výsledku. Ak skúsime spočítať počet prvkov nejakej nekonečnej množiny, počítanie nikdy neskončíme, a k porovnávaniu sa ani nedostaneme. To znamená, že potrebujeme novú všeobecnú metódu, ktorú by sme mohli aplikovať na porovnávanie mohutnosti množín bez ohľadu na to,



obr. 3.2

či sú to množiny konečné alebo nekonečné. Takto sa znova nachádzame na tej najhlbšej axiomatickej úrovni vedy. Našou úlohou je definovať pojem „nekonečna“ a dať presný význam vzťahu „menší alebo rovnako veľký ako“ pre mohutnosti dvoch množín.

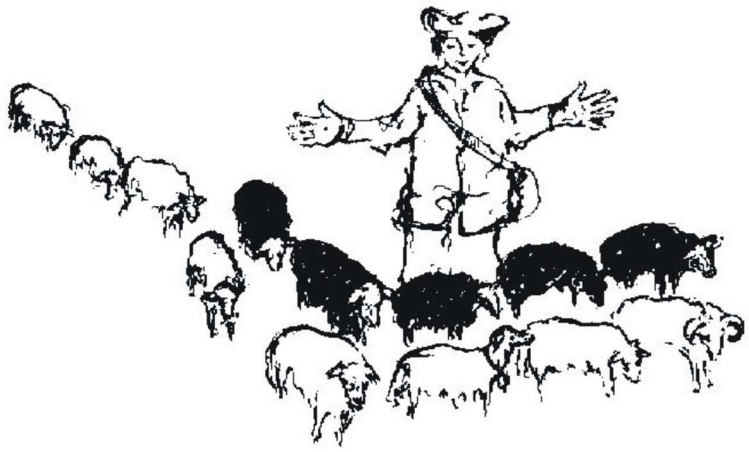
Tu sa necháme poučiť bačom. Nemusíme sa za to hanbiť, matematici to urobili tiež a okrem toho spájať múdrosť len so vzdelaním, by bolo veľkou chybou.

Náš bača má početnú čriedu oviec, v ktorej je veľa bielych a veľa čiernych oviec. Bača nechodil nikdy do školy, a preto napriek svojej múdrosti (ktorá ho drží vysoko v horách ďaleko od hektickej civilizácie), vie počítať len do troch. Bača chce zistiť, či má viac čiernych ako bielych oviec, alebo či je to naopak (obr. 3.2).

Ako to môže zistiť bez toho, aby ovce spočítal? Jednoducho. Vezme jednu bielu a jednu čiernu ovcu a vytvorí z nich jeden pár.

(biela ovca, čierna ovca).

Tento pár pošle z lúky do košiara. Potom vytvorí ďalší pár (biela ovca, čierna ovca) a pošle ho tiež do košiara (obr. 3.3). Takto bača pokračuje



obr. 3.3

dovtedy, pokiaľ na pasienku neostanú len ovce jedného druhu alebo tam ovce už nie sú. To znamená, že bača ukončí vytváranie párov, keď už na pasienku nemôže vytvoriť pár z jednej bielej a jednej čiernej ovce. Teraz uvažuje bača nasledujúco:

(i) Ak na pasienku neostali ovce, mám presne toľko bielych oviec koľko čiernych.

(ii) Ak mi na pasienku ostali len biele ovce, mám viac bielych ako čiernych oviec (obr. 3.3).

(iii) Ak mi na pasienku ostali len čierne ovce, mám viac čiernych ako bielych oviec.

Bačov záver (i) a párovanie ovcí si matematici zvolili za základný koncept na porovnávanie mohutnosti množín.

Definícia 3.1 Nech A a B sú dve množiny. **Párovanie** množín A a B je vytvorenie párov (a, b) tak, že platí:

(i) a patrí do A ($a \in A$) a b patrí do B ($b \in B$),

(ii) každý prvok z A sa nachádza práve v jednom páre párovania (nijaký prvok z A sa nenachádza v dvoch alebo viacerých pároch a nijaký z prvkov neostal nezaradený do páru (neostal na ocot),

(iii) každý prvok z B je druhým prvkom práve jedného páru v párovaní.

Pre každý pár (a, b) hovoríme, že ***a* a *b* sú spolu zosobášení**. Hovoríme,

že ***A* a *B* sú rovnako veľké** a píšeme

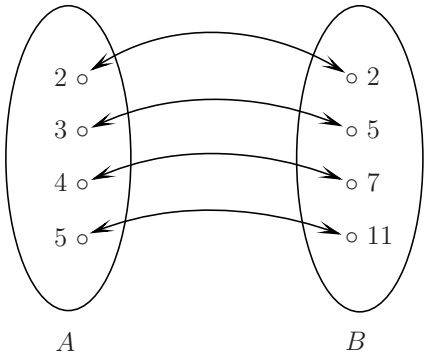
$$|\boldsymbol{A}| = |\boldsymbol{B}|\,,$$

ak existuje nejaké párovanie množín A a B. Hovoríme, že ***A* a *B* sú rôzne veľké** a píšeme

$$|\boldsymbol{A}| \neq |\boldsymbol{B}|\,,$$

ak neexistuje párovanie množín A a B.

Zamyslime sa napríklad nad množinami $A = \{2, 3, 4, 5\}$ a $B = \{2, 5, 7, 11\}$ na obr. 3.4.



obr. 3.4

Obrázok 3.4 znázorňuje párovanie:

$$(2, 2), (3, 5), (4, 7), (5, 11).$$

Každý prvok z A je práve v jednom páre ako prvý prvok (napríklad 4 z A je v treťom páre) a každý prvok páru z B je práve v jednom páre ako druhý prvok páru (napríklad 5 z B je v druhom páre). Inými slovami, každý prvok z A je zosobášený práve s jedným prvkom z B, každý prvok B je zosobášený práve s jedným prvkom z A, a teda nijaký prvok z A alebo B neostal slobodný. Vďaka tomu môžeme usúdiť, že platí

$$|\{2, 3, 4, 5\}| = |\{2, 5, 7, 11\}|\,.$$

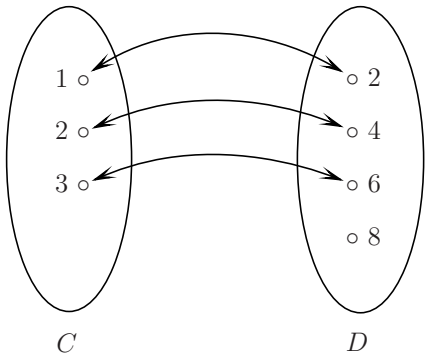
Samozrejme medzi A a B môžeme navrhnúť aj iné párovanie. Príklad iného možného párovania je:

$$(2, 11), (3, 7), (4, 5), (5, 2).$$

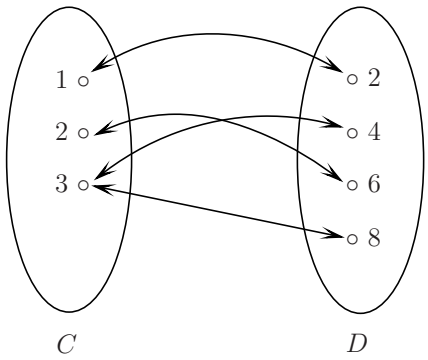
Úloha 3.1 a) Napíšte dve iné párovania množín $A = \{2, 3, 4, 5\}$ a $B = \{2, 5, 7, 11\}$.
b) Prečo nie je $(2, 2), (4, 5), (5, 11), (2, 7)$ párovaním množín A a B?

Podľa tohto konceptu sú množina A žien a množina B mužov rovnako mohutné, ak môžeme vydať všetky ženy a oženiť všetkých mužov tak, že nikto neostane na ocot[4].

Medzi množinami $C = \{1, 2, 3\}$ a $D = \{2, 4, 6, 8\}$ neexistuje párovanie, pretože každý pokus o párovanie sa skončí tým, že v množine D ostane jeden prvok naviac. To znamená, že $|D| \neq |C|$. Obrázok 3.5 ukazuje jeden neúspešný pokus o párovanie.



obr. 3.5



obr. 3.6

Obrázok 3.6 ukazuje iný neúspešný pokus o párovanie množín C a D. Tento pokus nevedie ku korektnému párovaniu, pretože prvok 3 z C je

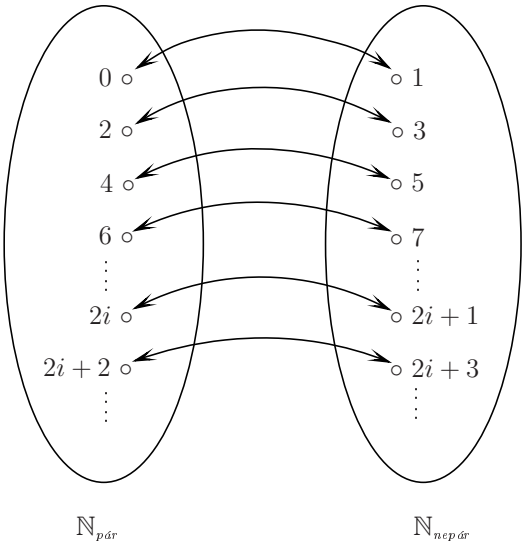
[4]Uvažujme len o svadbách párov s rôznym pohlavím.

v dvoch pároch $(3, 4)$ a $(3, 8)$, a teda je dvakrát ženatý.

Koncept párovania sme ale nezaviedli na to, aby sme porovnávali konečné množiny. To sme vedeli robiť už predtým aj bez tohto komplikovaného prístupu. Teraz sme si len overili, že v konečnom svete[5] náš koncept funguje správne. Keby to nebolo tak, bol by to chybný koncept a nemohli by sme ho používať. Pokúsme sa ho teraz aplikovať na nekonečné množiny. Predstavme si najprv množinu všetkých párnych čísiel a množinu všetkých nepárnych prirodzených čísiel. Tieto množiny sa zdajú byť rovnako veľké, takže sa to pokúsime naším konceptom porovnávania veľkosti množín aj zdôvodniť. Budeme párovať každé párne číslo $2i$ s o jednotku väčším nepárnym číslom $2i + 1$.

Na obrázku 3.7 vidíme, že takto dostávame nasledovných nekonečne veľa párov:

$$(0, 1), (2, 3), (4, 5), (6, 7), \ldots, (2i, 2i + 1), \ldots.$$



obr. 3.7

Presvedčme sa o tom, že táto postupnosť párov je korektným párovaním množín $\mathbb{N}_{pár}$ a $\mathbb{N}_{nepár}$. Nijaký prvok z $\mathbb{N}_{pár}$ alebo $\mathbb{N}_{nepár}$ nie je prvkom dvoch alebo viacerých párov (nie je viacnásobne ženatý). Ďalej vidíme, že nijaký prvok neostane slobodný. Pre každé párne číslo $2k$ z $\mathbb{N}_{pár}$ je toto číslo v páre $(2k, 2k+1)$. Pre každé nepárne číslo $2m+1$ z množiny $\mathbb{N}_{nepár}$

[5] Ak by koncept nefungoval v konečnom svete, zamietli by sme ho.

existuje v párovaní pár $(2m, 2m+1)$. Podľa nášho konceptu teda môžeme povedať, že $|\mathbb{N}_{párov}| = |\mathbb{N}_{nepár}|$.

Úloha 3.2 Dokážte, že $|\mathbb{Z}^+| = |\mathbb{Z}^-|$, pričom $\mathbb{Z}^+ = \{1, 2, 3, 4, \ldots\}$ a $\mathbb{Z}^- = \{-1, -2, -3, -4, \ldots\}$. Načrtnite párovanie aj graficky, podobne ako sme to urobili na obrázku 3.7.

Doteraz prebiehalo všetko podľa očakávania a naša argumentácia bola skutočne priezračná. Teraz prichádza niečo, čo nie je až tak jednoducho na prvý pokus stráviteľné. Pozrime sa na množiny:

$$\mathbb{N} = \{0, 1, 2, 3, \ldots\} \text{ a } \mathbb{Z}^+ = \{1, 2, 3, 4, \ldots\}.$$

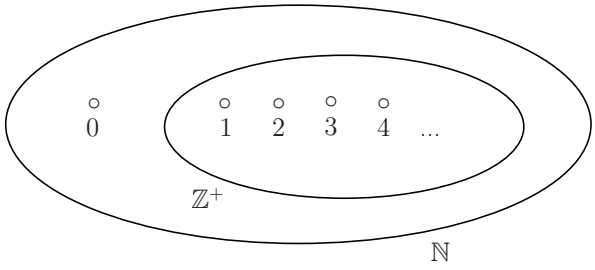
Všetky prvky množiny $\mathbb{Z}^+$ sú v $\mathbb{N}$, a teda platí:

$$\mathbb{Z}^+ \subseteq \mathbb{N},$$

čo znamená, že $\mathbb{Z}^+$ je **podmnožinou** množiny $\mathbb{N}$. Okrem toho platí, že prvok 0 je v $\mathbb{N}$, ale nie je v $\mathbb{Z}^+$. V takomto prípade hovoríme, že $\mathbb{Z}^+$ je **vlastná podmnožina** množiny $\mathbb{N}$ a píšeme $\mathbb{Z}^+ \subset \mathbb{N}$. Pojem „$A$ je vlastná podmnožina B“ znamená, že A je časťou B, ale nie celým B. Názorne túto situáciu vidíme pre:

$$\mathbb{Z}^+ \subset \mathbb{N}.$$

na obrázku 3.8. Množina $\mathbb{Z}$ je v $\mathbb{N}$ úplne obsiahnutá, ale nie je to celá množina $\mathbb{N}$, lebo $0 \in \mathbb{N}$ a $0 \notin \mathbb{Z}^+$.



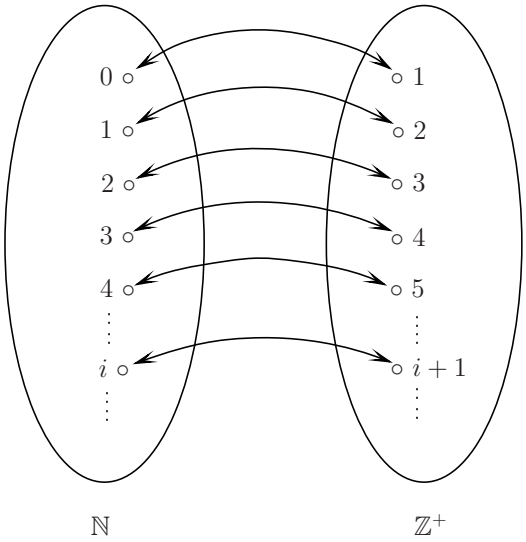
obr. 3.8

Napriek tomu teraz tvrdíme, že platí:

$$|\mathbb{N}| = \left|\mathbb{Z}^+\right|,$$

teda, že tieto dve nekonečná $|\mathbb{N}|$ a $\left|\mathbb{Z}^+\right|$ sú rovnako veľké. Odôvodníme to nasledujúcim párovaním:

$$(0, 1), (1, 2), (2, 3), \ldots, (i, i+1), \ldots ,$$



obr. 3.9

ktoré je znázornené na obrázku 3.9.

Jasne vidíme, že prvky z množín $\mathbb{N}$ a $\mathbb{Z}^+$ sú korektne popárované (zosobášené). Nijaký prvok neostal na ocot (slobodný). To znamená, že $\mathbb{N}$ nie je väčšie ako $\mathbb{Z}^+$, aj keď očividne má $\mathbb{N}$ viacej prvkov ako $\mathbb{Z}^+$. Náš výsledok hovorí v podstate, že platí:

$$\infty + 1 = \infty.$$

Inými slovami, pripočítanie jednotky k nekonečnu nevedie k vzniku väčšieho nekonečného čísla. V tejto formulácii to už nie je až také prekvapujúce. Čože je 1 v porovnaní s nekonečnom? Také maličké nič, ktoré smieme prehliadnuť. Táto zdanlivo prekvapujúca kombinácia vlastností množín

$$\mathbb{Z}^+ \subset \mathbb{N} \text{ (obr. 3.8) a } \left|\mathbb{Z}^+\right| = |\mathbb{N}| \text{ (obr. 3.9)}$$

je základom, na ktorom môžeme postaviť definíciu nekonečna. Na objavenie tejto definície matematici potrebovali stáročia a na jej pochopenie a akceptovanie desaťročia. Táto definícia poskytuje presne to, čo sa od definície nekonečna očakáva. Na základe tejto definície sme schopní rozlišovať konečné množiny od nekonečných množín.

Definícia 3.2 Množina A je **nekonečná** vtedy a len vtedy, ak obsahuje vlastnú podmnožinu B takú, že platí:

$$|A| = |B|.$$

Inými slovami to môžeme sformulovať takto:

> *„Nejaký objekt je nekonečný práve vtedy, ak obsahuje nejakú vlastnú časť, ktorá je rovnako veľká ako celý objekt. “*

Teraz môžete namietať: *„Toto zašlo už priďaleko. S takýmto niečím nemôžem súhlasiť. Ako môže byť nejaká vlastná časť nejakého celku rovnako veľká ako samotný celok? Také dačo predsa neexistuje!“*

Teší ma, že takto rozmýšľate. Práve preto je táto definícia vhodná. V reálnom svete, v ktorom je všetko konečné, nemôže byť nijaká časť celku rovnako veľká ako samotný celok. Na tomto sa asi zhodneme. Teda nijaký reálny (konečný) objekt túto zvláštnu vlastnosť nemá. Takže v súlade s definíciou 3.2 nie sú tieto objekty nekonečné, teda sú konečné. Ale v hypotetickom svete nekonečna je nielen možné, ale je dokonca povinnosťou mať túto zvláštnu vlastnosť. A táto vlastnosť je presne to, čo potrebujeme na odlíšenie nekonečna od konečných objektov. Objekt, ktorý má vlastnosť podľa definície 3.2 je nekonečný a objekty, ktoré túto vlastnosť nemajú, sú konečné. Takto pomocou definície 3.2 môžeme triediť objekty na konečné a nekonečné a to je práve to, kvôli čomu sme sa snažili nájsť definíciu nekonečna.

Aby sme lepšie pochopili túto čudesnú a pritom predsa len charakteristickú vlastnosť nekonečných objektov, uvádzame nasledujúce dva príklady.

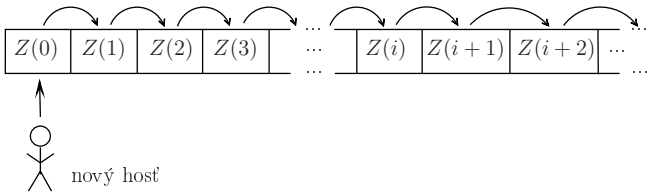
Príklad 3.1 Hotel Hilbert

Hilbertov hotel je rozprávkový hotel. Má nekonečne veľa jednoposteľových izieb. Tieto izby sú očíslované resp. pomenované takto:

$$Z(0), Z(1), Z(2), Z(3), \ldots, Z(i), \ldots .$$

V okamihu príchodu nového hosťa sú všetky izby obsadené, v každej býva práve jeden hosť. Nový hosť sa pýta recepčného: „Máte pre mňa voľnú izbu?“ „Samozrejme“, odpovedá recepčný a ubytuje nového hosťa nasledujúcim spôsobom. Požiada každého hosťa v hoteli, aby sa presťahoval do izby s poradovým číslom o jednotku vyšším ako má izba, kde je teraz. Hosť z izby $Z(0)$ sa takto presťahuje do izby $Z(1)$, hosť z izby $Z(1)$ sa ubytuje v izbe $Z(2)$ a tak ďalej. Vo všeobecnosti môžeme povedať, že hosť z izby $Z(i)$ sa presťahuje do izby $Z(i+1)$. Týmto spôsobom sa uvoľní izba $Z(0)$ a recepčný túto izbu dá novému hosťovi (obr. 3.10).

Vidíme, že po takomto presťahovaní je izba $Z(0)$ voľná a každý z hotelových hostí má pre seba vlastnú izbu. Matematici by odôvodnili toto



obr. 3.10

tvrdenie takto: Pretože hosť z izby $Z(0)$ sa odsťahoval, je očividné, že táto izba sa uvoľnila a ostala po sťahovaní prázdna. Potrebujeme ešte ukázať, že po sťahovaní má každý hosť vlastnú izbu. Nech H je ľubovoľný hosť. Tento hosť býval pred sťahovaním v konkrétnej izbe. Nech $Z(n)$ je izba hosťa H pred sťahovaním. Podľa pokynu recepčného opustí H izbu $Z(n)$ a presťahuje sa do izby $Z(n + 1)$. To H môže urobiť, pretože pôvodný obyvateľ izby $Z(n + 1)$ sa presťahoval do izby $Z(n + 2)$. Vďaka tomu je po sťahovaní hosť H jediným obyvateľom izby $Z(n + 1)$. Pretože táto úvaha platí pre každého hosťa ubytovaného v hoteli, sú po sťahovaní všetci hostia vo vlastných jednoposteľových izbách.

Toto riešenie ukazuje, prečo bolo aktuálne nekonečno dlhý čas v matematike paradoxom[6]. Nekonečný Hilbertov Hotel je aktuálne nekonečno. Také niečo možno znázorniť len znázornením konečnej časti a troma bodkami. Preto nie je možné naraz sledovať úspešné sťahovanie nekonečne veľa hostí. Ale každého jedného hosťa môžeme pozorovať jednotlivo a presvedčiť sa, že sťahovanie funguje.

Tento paradox sa podarilo rozriešiť[7] až keď vedci rozpoznali, že sa konečné od nekonečného odlišuje práve v tom, že nekonečné objekty majú časti rovnako veľké ako celok. Dôležité je všimnúť si, že sťahovanie v Hilbertovom hoteli zodpovedá párovaniu množiny $\mathbb{N}$ predstavujúcej množinu hostí a množiny $\mathbb{Z}^+$ predstavujúcej množinu izieb, začínajúc izbou $Z(1)$. □

Úloha 3.3 a) Do Hilbertovho hotela prídu traja noví hostia. Tak ako predtým je hotel plne obsadený. Prevezmite rolu recepčného a ubytujte nových troch hostí bez toho, aby ste niektorého už z ubytovaných hostí poslali preč. Podľa možnosti ale nežiadajte bývajúcich hostí, aby sa presťahovali trikrát. Usilujte sa vyriešiť situáciu jedným sťahovaním každého z hostí.

b) Do plne obsadeného Hilbertovho hotela prichádza nový hosť a bezpodmienečne požaduje izbu $Z(7)$. Ako mu môže recepčný vyhovieť?

[6] zdanlivo rozporuplná alebo nevysvetliteľná situácia

[7] teda už nie je viac paradoxom

Nasledujúci príklad pochádza z fyziky. Vymysleli si ho fyzici, ako liek na liečenie depresií, ktoré spôsobili sami tým, že vypočítali, aká maličká a bezvýznamná je naša zemeguľa (a v tom istom zmysle aj ľudstvo) v porovnaní s obrovským vesmírom.

Príklad 3.2 Predstavme si zemeguľu ako nekonečnú množinu bodov, ktoré nemajú rozmer (veľkosť), teda môžu ležať ľubovoľne blízko vedľa seba. Tak isto, ako množinu bodov si môžeme predstaviť aj vesmír. Kvôli zjednodušeniu budeme všetko znázorňovať dvojrozmerne namiesto trojrozmerne. Vesmír je ľubovoľne veľký list papiera a zemeguľa je malý kruh na papieri (obr. 3.11). Keby mal niekto problém s tým predstaviť si zemeguľu ako nekonečnú množinu bodov, poznamenajme, že už len konečná úsečka medzi 0 a 1 na reálnej osi obsahuje nekonečný počet bodov. Každé racionálne číslo (zlomok) medzi číslami 0 a 1 si môžeme predstaviť ako bod na úsečke medzi číslami 0 a 1. A už vieme, že medzi nulou a jednotkou existuje nekonečne veľa rôznych racionálnych čísiel. Ukázali sme to tým, že sme generovali nekonečné postupnosti menších a menších kladných racionálnych čísiel pri neúspešnom pokuse nájsť najmenšie kladné racionálne číslo. Iné odôvodnenie nekonečného počtu čísiel medzi nulou a jednotkou je založené na dôkaze nasledujúceho tvrdenia.

> Medzi ľubovoľnými dvoma rôznymi racionálnymi číslami a a b, $a < b$, leží nekonečne veľa racionálnych čísiel.

Ako prvé číslo medzi a a b uvažujeme číslo $c_1 = \frac{a+b}{2}$, ktoré je priemernou hodnotou a a b. Ako druhé vezmeme číslo $c_2 = \frac{c_1+b}{2}$, čo je priemer medzi c_1 a b. Vo všeobecnosti číslo

$$c_i = \frac{c_{i-1} + b}{2}$$

je priemernou hodnotou čísiel c_{i-1} a b. A teda leží medzi $a = 0$ a $b = 1$. je zrejmé, že takéto generovanie čísiel medzi a a b sa nikdy neskončí. Ak zvolíme konkrétne $a = 0$ a $b = 1$, tak dostávame nekonečnú postupnosť

$$\frac{1}{2}\ ,\ \frac{3}{4}\ ,\ \frac{7}{8}\ ,\ \frac{15}{16}\ ,\ \cdots$$

rôznych zlomkov medzi nulou a jednotkou.

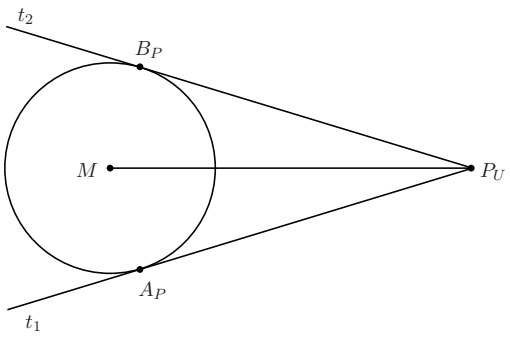
Poďme ale konečne k tomu, čo nám chceli povedať fyzici. Fyzici mali výčitky svedomia, že ukázali, aká je zemeguľa maličká v porovnaní s obrovským vesmírom. Predstava o ničotnosti ľudstva v tomto porovnaní spôsobovala mnohým ľuďom depresívne stavy. V snahe zachrániť situáciu dokázali fyzici takéto tvrdenie: Všetky body vesmíru mimo zemegule sa

dajú spárovať s bodmi vo vnútri zemegule. Toto tvrdenie má hneď dve pozitívne optimistické interpretácie:

(i) Počet bodov zemegule je rovný počtu bodov vesmíru okolo zemegule.

(ii) Všetko, čo sa odohráva v obrovskom vesmíre sa dá zobraziť na zemeguli, teda odzrkadliť to v zmenšenom merítku.

Pohľadajme teraz párovanie bodov vnútri a mimo zemegule. Každému bodu P_U mimo zemegule priradíme bod P_E vo vnútri zemegule nasledujúcim spôsobom.



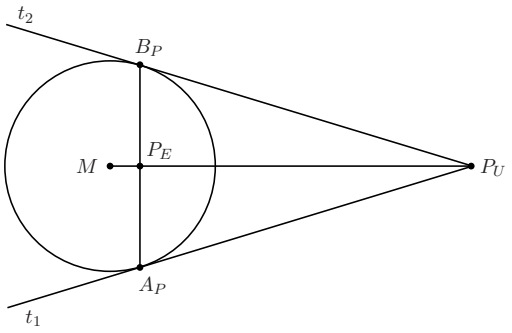
obr. 3.11

Najprv spojme priamkou bod P_U so stredom zemegule M (obr. 3.11). Naším cieľom je určiť vo vnútri zemegule na tejto priamke bod P_E a tak vytvoriť pár (P_U, P_E). Aby sme to dosiahli, z bodu P_U narysujeme dotyčnice zeme t_1 a t_2. Dotyčnica kružnice je priamka, ktorá sa kružnice dotýka práve v jednom bode. Body, v ktorých t_1 a t_2 sa dotýkajú zemegule, označíme A_P a P_B (obr. 3.11). Teraz spojíme body A_P a B_P priamkou a dostaneme úsečku A_PB_P (obr. 3.12). Úsečky MP_U a A_PB_P sa pretínajú práve v jednom bode a to je nami hľadaný bod P_E (obr. 3.12). Takto sme vytvorili pár bodu P_U z vesmíru s bodom zemegule P_E.

Teraz ešte musíme ukázať, že takýmto spôsobom párovania vždy priradíme dvom rôznym bodom vesmíru mimo zemegule P_U a $P_U^{'}$ dva rôzne body zemegule. Týmto dokážeme, že skutočne ide o korektné párovanie.

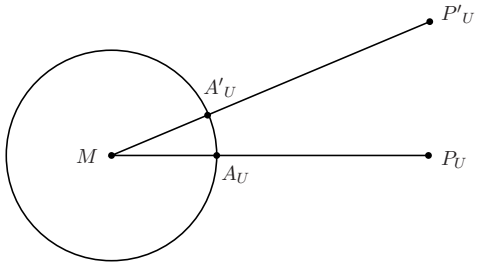
Budeme rozlišovať dve možnosti.

(i) Body M, P_U a $P_U^{'}$ neležia na tej istej priamke. Situácia je znázornená na obrázku obr. 3.13. Vieme, že P_E musí ležať na úsečke MP_U a $P_E^{'}$ na úsečke $MP_U^{'}$. Vzhľadom na to, že tieto úsečky nemajú ok-

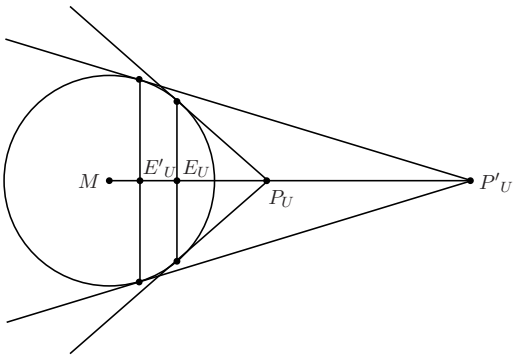


obr. 3.12

rem bodu M iný spoločný bod a bod M nie je priradený nijakému bodu mimo zemegule, body P_E a $P_E^{'}$ sú rôzne bez ohľadu na to, kde na úsečkách MP_U a $MP_U^{'}$ ležia.



obr. 3.13: E_U leží na MA_U a $E_U^{'}$ leží na $MA_U^{'}$, a preto sú body E_U a $E_U^{'}$ rozdielne.



obr. 3.14

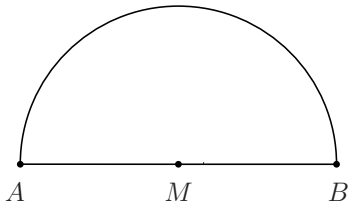
(ii) Všetky tri body M, P_U a $P_U^{'}$ ležia na jednej priamke (obr. 3.14).

Takže na jednej priamke musia ležať aj body E_U a E_U'. V tomto prípade realizujeme konštrukciu bodov P_U a P_U' tak isto ako na obrázku obr. 3.12. Na obrázku obr. 3.14 priamo vidíme, že body E_U a E_U' sú rôzne.

Ukázali sme, že bez ohľadu na to, koľkokrát je vesmír väčší ako naša maličká zemeguľa, počet bodov zemegule a vesmíru je rovnaký. □

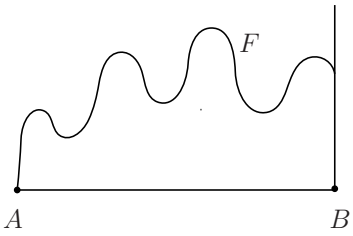
Úloha 3.4 Doplňte obrázok 3.13 tak, že určíte presne polohu bodov P_E a P_E'.

Úloha 3.5 Pozrime sa na polkružnicu na obrázku 3.15 a úsečku AB, ktorej dĺžka je rovnako veľká ako priemer kružnice. Geometricky a prípadne aj výpočtom zdôvodnite, prečo má úsečka AB presne taký istý počet bodov, ako zobrazená polkružnica.



obr. 3.15

Úloha 3.6 Pozrime sa na graf funkcie F na obrázku 3.16 a úsečku AB. Prečo má graf funkcie a úsečka AB presne taký istý počet bodov?



obr. 3.16

Ak je vám pri pokuse akceptovať prezentovaný koncept nekonečnosti ešte stále nevoľno, nenechajte sa tým príliš znepokojiť. Svetoznámi matematici devätnásteho storočia potrebovali dlhé roky nielen na objavenie tohto konceptu, ale i na jeho uznanie v matematickej obci. Niektorí známi matematici až do svojej smrti odmietali predstavenú definíciu nekonečna a porovnávanie veľkosti nekonečien. Niektorí dokonca označovali tieto myšlienky pre matematiku za „kacírske“ a bojovali proti ich publikovaniu

aj prostriedkami, ktoré do vedy nepatria. Preto si pokojne dožičte čas a opakovane sa s definíciou nekonečna konfrontujte. Len po opakovanom konfrontovaní dokážeme pochopiť, prečo bola matematikmi prijatá ako axióma práve táto definícia nekonečna a prečo ju dnes matematici pokladajú nielen že za dôveryhodnú, ale ani si dokonca nevedia predstaviť akceptovateľnú alternatívu na definovanie a zaobchádzanie s nekonečnom.

Skúsme v krátkosti porozmýšľať nad alternatívnym konceptom porovnávania nekonečien, ktorý občas poslucháči pri prvom stretnutí s nekonečnom navrhujú ako prijateľnejšiu možnosť. Ak platí:

$$A \subset B$$

(t.j. ak A je vlastná podmnožina množiny B), tak platí

$$|A| < |B|\,.$$

Tento pokus očividne odmieta kľúčovú definíciu nekonečna, ktorá požaduje, aby nejaká časť celku bola rovnako veľká ako samotný celok. Ale tento pokus má dve slabiny. Po prvé umožňuje porovnávať len také dve množiny, kde jedna je podmnožinou druhej. Takže tento prístup nám neumožňuje porovnávať rôzne množiny, napríklad $\mathbb{Z}^- = \{-1, -2, -3, \ldots\}$ a $\mathbb{Z}^+ = \{1, 2, 3, \ldots\}$. Musíme teda hľadať a poskytnúť aj nejakú možnosť na porovnávanie rôznych množín. Poslucháči väčšinou navrhnú zobraziť jednu z týchto množín pomocou párovania na nejakú podmnožinu tej druhej množiny a potom realizovať porovnanie. Teraz vám ukážeme, že tento prístup si nemôžeme dovoliť, pretože vedie k sporu, lepšie povedané k evidentne nezmyselnému tvrdeniu. To nezmyselné tvrdenie je:

$$|\mathbb{N}| < |\mathbb{N}|,$$

teda že $\mathbb{N}$ je menšie ako samotné $\mathbb{N}$. Ukážeme, že tento nezmysel je dôsledkom navrhovaného alternatívneho konceptu porovnávania nekonečien. Pomocou párovania sme ukázali, že platí:

$$|\mathbb{N}| = |\mathbb{Z}^+|. \tag{3.1}$$

Pretože $\mathbb{Z}^+ \subset \mathbb{N}$, platí podľa alternatívneho konceptu porovnávania nekonečien:

$$|\mathbb{Z}^+| < |\mathbb{N}|. \tag{3.2}$$

Ak napíšeme vzťahy (3.1) a (3.2) za sebou, dostávame

$$|\mathbb{N}| = |\mathbb{Z}^+| < |\mathbb{N}|$$

a teda $|\mathbb{N}| < |\mathbb{N}|$. Takto vidíme, že koncept porovnávania veľkosti množín pomocou párovania nie je zlučiteľný s predstavou, že každá vlastná podmnožina musí byť menšia ako celá množina.

Úloha 3.7 (tvrdý oriešok) Predstavte si, že vám niekto navrhne nasledujúcu definíciu na porovnávanie mohutností dvoch množín A a B.

Mohutnosť množiny A je menšia ako mohutnosť množiny B, $|A| < |B|$, práve vtedy, ak existuje nejaká vlastná podmnožina C množiny B ($C \subset B$), že prvky množín A a C možno popárovať (existuje korektné párovanie medzi A a C).

Ukážte, že táto definícia vedie k sporu (ako dôsledok tejto definície dostanete $|\mathbb{N}| < |\mathbb{N}|$).

Prečo o tejto axióme matematiky toľko diskutujeme a venujeme toľké úsilie na sprostredkovanie jej pochopenia? Ako už asi tušíte, jednoducho preto, že táto axióma je len prvým prekvapením na našej ceste na hranicu automaticky riešiteľného, ktorým ideme konkurovať Alici v krajine zázrakov. Práve sme v istom zmysle dokázali, že $\infty = \infty+1$, keď $\infty = |\mathbb{N}|$ a v podstate aj naznačili, že pre ľubovoľné konečné číslo c platí $\infty = \infty+c$. Príklad 3.2 a za ním nasledujúce cvičenia už dokonca naznačujú, že pre ľubovoľné konečné číslo (ľubovoľnú konštantu) c platí:

$$\infty = c \cdot \infty.$$

Porovnajme veľkosti množiny $\mathbb{N}$ a množiny

$$\mathbb{N}_{pár} = \{0, 2, 4, 6, \ldots\} = \{2i \mid i \in \mathbb{N}\}$$

všetkých párnych prirodzených čísel. Na prvý pohľad by sme odhadli, že $\mathbb{N}$ obsahuje dvakrát toľko čísel, ako $\mathbb{N}_{pár}$. Napriek tomu (obr. 3.17) môžeme prvky množín $\mathbb{N}$ a $\mathbb{N}_{pár}$ popárovať nasledujúcim spôsobom:

$$(0,0), (1,2), (2,4), (3,6), \ldots, (i, 2i), \ldots.$$

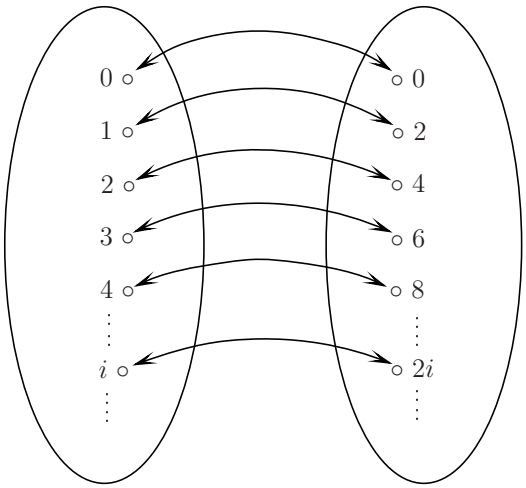
Pozorujeme, že každý prvok oboch množín je zosobášený práve raz. Ako dôsledok dostávame:

$$|\mathbb{N}| = |\mathbb{N}_{pár}|.$$

Tento prekvapujúci výsledok hovoriaci, že

$$2 \cdot \infty = \infty$$

si môžeme opätovne potvrdiť príbehom z Hilbertovho hotela.



obr. 3.17

Príklad 3.3 Rozmýšľajme znova o Hilbertovom hoteli s nekonečne veľa izbami

$$Z(0), Z(1), Z(2), \dots \ ,$$

ktoré sú všetky obsadené. V tejto situácii príde nekonečný autobus s miestami na sedenie

$$B(0), B(1), B(2), \dots \ ,$$

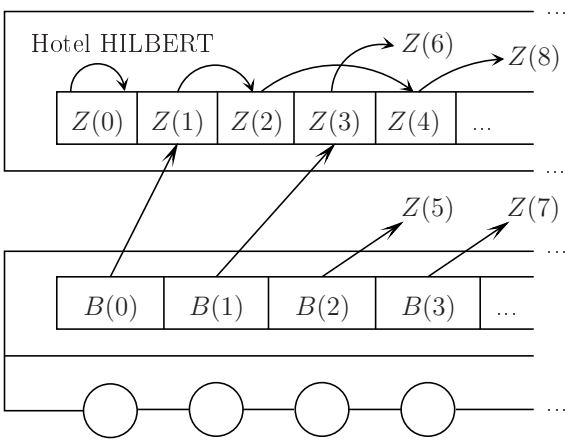
ktoré sú všetky obsadené[8]. Vodič autobusu sa pýta recepčného, či je možné ubytovať všetkých pasažierov autobusu. Recepčný, ako zvyčajne odpovedá: „To nie je problém“ a urobí nasledujúce:

Požiada každého hosťa, aby sa presťahoval zo svojej izby $Z(i)$ do izby $Z(2i)$, ako je to znázornené v hornej časti obrázku 3.18. Po presťahovaní má znova každý hosť vlastnú izbu a všetky izby $Z(2i+1)$ s nepárnymi číslami $1, 3, 5, 7, \dots, 2i+1, \dots$ sú voľné. Teraz má recepčný už len úlohu nájsť párovanie medzi voľnými izbami a sedadlami v autobuse. Hosťovi zo sedadla $B(0)$ pridelí izbu $Z(1)$, hosťovi zo sedadla $B(1)$ izbu $Z(3)$, atď. Vo všeobecnosti pridelí cestujúcemu sediacemu na mieste $B(i)$ izbu $Z(2i+1)$, ako je to znázornené na obrázku 3.18. Takto dostávame párovanie:

$$(B(0), Z(1)), (B(1), Z(3)), (B(2), Z(5)), \dots, (B(i), Z(2i+1)), \dots$$

medzi uvoľnenými izbami s nepárnymi číslami a sedadlami nekonečného autobusu.

[8]Na každom sedadle sedí práve jeden pasažier.



obr. 3.18

□

Úloha 3.8 a) Hilbertov hotel je poloprázdny. Všetky izby s párnymi číslami $Z(0), Z(2), Z(4), \ldots$ sú obsadené a všetky izby s nepárnymi číslami sú voľné. V tejto situácii prídu dva nekonečne dlhé autobusy B_1 a B_2, ktorých miesta (sedadlá) sú označené takto:

$$B_1(0), B_1(1), B_1(2), B_1(3), \ldots$$

$$B_2(0), B_2(1), B_2(2), B_2(3), \ldots.$$

Ako môže postupovať recepčný, aby poskytol ubytovanie všetkým pasažierom v týchto dvoch autobusoch?

b) Hilbertov hotel je plne obsadený a prídu tri nekonečné autobusy, ktorých sedadlá sú očíslované postupne všetkými prirodzenými číslami. Ako možno ubytovať všetkých cestujúcich?

Úloha 3.9 Ukážte pomocou párovania, že platí:

$$|\mathbb{Z}| = |\mathbb{N}|,$$

pričom $\mathbb{Z} = \{\ldots, -3, -2, -1, 0, 1, 2, 3, \ldots\}$ je množina všetkých celých čísel.

Úloha 3.10 (tvrdý oriešok) Nech $[a, b]$ označuje množinu všetkých bodov (všetkých reálnych čísel) na reálnej osi medzi číslami a a b.

a) Ukážte platnosť rovnosti:

$$|[0, 1]| = |[1, 10]|.$$

Vyskúšajte to geometricky podobne ako v príklade 3.2.

b) Ukážte aritmeticky platnosť rovnosti:

$$|[0,1]| = |[0,100]|,$$

t.j. nájdite vhodnú funkciu f takú, aby pre $i \in [0,100]$ bolo $(f(i), i)$ párovaním množín

$$|[0,1]| = |[0,100]|.$$

Úloha 3.11 (tvrdý oriešok) Predstavme si, že Hilbertov hotel je úplne prázdny, teda v ňom nie je nik ubytovaný. Vzhľadom na to, že predchádzajúce stratégie ubytovávania hostí vyžadovali neustále sťahovanie sa z izby do izby, hrozí, že záujem o ubytovanie v hoteli výrazne poklesne. Nikto nemá záujem pri každom príchode nového hosťa meniť izbu. Evidentne recepčný potrebuje nejakú novú ubytovávaciu stratégiu, pomocou ktorej môže ubytovávať hostí bez toho, aby sa čo len jeden niekedy presťahoval. Toto ubytovanie musí fungovať bez ohľadu na to, koľko konečných alebo nekonečných autobusov, kedy a v akom poradí príde do hotela. Viete recepčnému pomôcť?

Pozorujeme, že dokázať pre nejakú množinu A rovnosť

$$|\mathbb{N}| = |A|$$

neznamená nič iné, ako očíslovať všetky prvky množiny A. Párovanie medzi $\mathbb{N}$ a A priradí jednoducho každému prvku z A nejaké prirodzené číslo (poradie) z $\mathbb{N}$. Toto číslo môžeme pokladať za poradové číslo prvku z A. Ak je napríklad (3, Jan) pár z nejakého párovania medzi $\mathbb{N}$ a A, tak môžeme prvok Jan z množiny A označiť za tretí prvok tejto množiny. Naopak, každé očíslovanie prvkov nejakej množiny A nám poskytne párovanie medzi $\mathbb{N}$ a A. Pármi v párovaní sú:

$$(\text{poradie prvku } a, a),$$

pre každý prvok a z A. Pojem **očíslovania** nám pre ľubovoľnú množinu A ponúka názornú argumentáciu na dokazovanie rovnosti $|\mathbb{N}| = |A|$, to znamená dokazovanie skutočnosti, že A má rovnako veľa prvkov ako $|\mathbb{N}|$.

Pomocou párovania

$$(0,0), (1,1), (2,-1), (3,2), (4,-2), (5,3), (6,-3), \ldots$$

množín $\mathbb{N}$ a $\mathbb{Z}$ priraďujeme prvkom množiny $\mathbb{Z}$ poradie:

$$0, 1, -1, 2, -2, 3, -3, \ldots.$$

Takto je 0 0-tým prvkom, 1 je prvým prvkom, -1 je druhým prvkom v $\mathbb{Z}$, atď. Vidíme, že usporiadanie prvkov $\mathbb{Z}$ do radu, dané nejakým párovaním a jemu zodpovedajúcim očíslovaním, nemá nič spoločné s významom

alebo hodnotou prvkov zo $\mathbb{Z}$. Hodnoty prvkov zo $\mathbb{Z}$ v našom poradí nie sú ani rastúce, ani klesajúce. Rôzne párovania množín $\mathbb{N}$ a A môžu viesť k rôznym očíslovaniam prvkov z A, a tým aj k rôznym poradiam.

Úloha 3.12 Nájdite nejaké iné párovanie medzi množinami $\mathbb{N}$ a $\mathbb{Z}$, ktoré vedie k inému usporiadaniu prvkov $\mathbb{Z}$ do radu.

Úloha 3.13 Dokážte platnosť rovnosti

$$|\mathbb{N}| = |\mathbb{N}_{quad}|,$$

kde $\mathbb{N}_{quad} = \{i^2 \mid i \in \mathbb{N}\} = \{0, 1, 4, 9, 16, 25, \ldots\}$ je množina všetkých štvorcov prirodzených čísiel. Aké poradie prvkov z $\mathbb{N}_{quad}$ dostanete na základe vami navrhnutého párovania množín $\mathbb{N}$ a $\mathbb{N}_{quad}$?

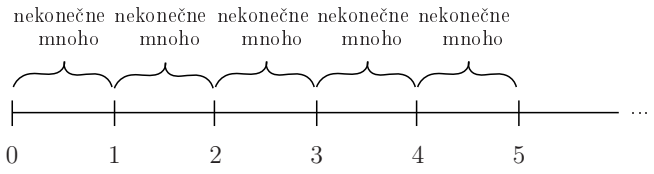
Zodpovedanie nasledujúcej otázky zvýši stupeň obťažnosť našich úvah. V akom vzťahu sú nekonečná $|\mathbb{N}|$ a $|\mathbb{Q}^+|$?

$$\mathbb{Q}^+ = \left\{ \frac{p}{q} \mid p, q \in \mathbb{Z}^+ \right\}$$

je množina všetkých kladných racionálnych čísiel (kladných zlomkov). Už v predchádzajúcom texte sme videli, že nekonečne veľa racionálnych čísiel môžeme nájsť pomocou opakovaného počítania priemeru medzi ľubovoľnými dvoma rôznymi racionálnymi číslami a a b, $a < b$. Ak rozdelíme kladnú reálnu polos na nekonečne veľa častí $[0, 1], [1, 2], [2, 3], \ldots$, ako na obrázku 3.19, vyzerá, že mohutnosť $|\mathbb{Q}^+|$ je

$$\infty \cdot \infty = \infty^2,$$

pretože každá z tých nekonečne veľa častí osi obsahuje nekonečne veľa racionálnych čísiel.



obr. 3.19

Z tohto pohľadu nevyzerá veľmi sľubné očakávať platnosť rovnosti $|\mathbb{N}| = |\mathbb{Q}^+|$, a tak skôr tipujeme, že $|\mathbb{Q}^+|$ je väčšie nekonečno, ako nekonečno $|\mathbb{N}|$. Prirodzené čísla $0, 1, 2, 3, \ldots$ ležia na kladnej polosi veľmi riedko a medzi

každými dvoma číslami i a $i+1$, leží nekonečne veľa racionálnych čísiel. Naviac, uvedomujeme si, že existencia párovania medzi množinami $\mathbb{N}$ a $\mathbb{Q}^+$ by automaticky znamenala existenciu očíslovania prvkov množiny $\mathbb{Q}^+$, a tým aj ich usporiadania do radu. Čísla z $\mathbb{Q}^+$ podľa veľkosti určite nemôžeme zoradiť, pretože (ako sme už uviedli) neexistuje najmenšie kladné racionálne číslo[9].

Napriek tomuto prvému dojmu ukážeme, že platí,

$$|\mathbb{N}| = |\mathbb{Q}^+|$$

a tak, že v istom zmysle platí:

$$\infty \cdot \infty = \infty.$$

Najprv si pripomenieme, že v množine $\mathbb{Z} = \{\ldots, -3, -2, -1, 0, 1, 2, 3, \ldots\}$ tiež neexistuje najmenšie číslo, a napriek tomu sa prvky z $\mathbb{Z}$ dajú očíslovať v poradí

$$0, -1, 1, -2, 2, -3, 3, \ldots.$$

Myšlienka ako párovať množiny $\mathbb{N}$ a $\mathbb{Q}^+$ je založená na napísaní si všetkých kladných čísel (zlomkov) na, v oboch rozmeroch, nekonečný list papiera. Matematici medzi nami by povedali, že zobrazíme všetky prvky $\mathbb{Q}^+$ na pozície nekonečnej dvojrozmernej matice znázornenej na obrázku 3.20. Vieme, že každé kladné racionálne číslo môžeme zapísať v tvare zlomku ako

$$\frac{p}{q},$$

pričom p a q sú kladné celé čísla. Nekonečný list papiera rozdeľme na nekonečne veľa stĺpcov a očíslujeme riadky zhora nadol a stĺpce zľava doprava číslami

$$1, 2, 3, 4, 5, \ldots.$$

Na pozíciu nekonečného listu papiera, kde sa pretína i-ty riadok a j-ty stĺpec, napíšeme zlomok

$$\frac{i}{j}.$$

Konečná časť takto popísaného nekonečného listu papiera (nekonečnej matice) je na obrázku 3.20.

Nepochybujeme o tom, že na tomto nekonečnom liste papiera sa nachádzajú všetky kladné zlomky (racionálne čísla). Ak hľadáme ľubovoľný

[9] Pre ľubovoľné malé kladné racionálne číslo a môžeme pomocou delenia dvomi vytvoriť zlomok $a/2$, ktorý je menší ako a.

	1	2	3	4	5	6	...
1	$\frac{1}{1}$	$\frac{1}{2}$	$\frac{1}{3}$	$\frac{1}{4}$	$\frac{1}{5}$	$\frac{1}{6}$	⋯
2	$\frac{2}{1}$	$\frac{2}{2}$	$\frac{2}{3}$	$\frac{2}{4}$	$\frac{2}{5}$	$\frac{2}{6}$	⋯
3	$\frac{3}{1}$	$\frac{3}{2}$	$\frac{3}{3}$	$\frac{3}{4}$	$\frac{3}{5}$	$\frac{3}{6}$	⋯
4	$\frac{4}{1}$	$\frac{4}{2}$	$\frac{4}{3}$	$\frac{4}{4}$	$\frac{4}{5}$	$\frac{4}{6}$	⋯
5	$\frac{5}{1}$	$\frac{5}{2}$	$\frac{5}{3}$	$\frac{5}{4}$	$\frac{5}{5}$	$\frac{5}{6}$	⋯
6	$\frac{6}{1}$	$\frac{6}{2}$	$\frac{6}{3}$	$\frac{6}{4}$	$\frac{6}{5}$	$\frac{6}{6}$	⋯
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋱

obr. 3.20

zlomok p/q, okamžite vieme, že tento zlomok sa nachádza na priesečníku p-teho riadku a j-tého stĺpca. Máme skôr iný problém. Niektoré[10] kladné zlomky sa v tabuľke nachádzajú viackrát, dokonca nekonečne veľakrát. Napríklad číslo 1 sa dá zapísať, ako

$$\frac{1}{1}, \frac{2}{2}, \frac{3}{3}, \frac{4}{4}, \ldots,$$

teda na našom nekonečnom liste sa nachádza nekonečne veľakrát na všetkých pozíciách diagonály. Číslo $1/2$ tu nájdeme zapísané tiež nekonečne veľakrát, pretože jednu polovicu možno zapísať v tvare zlomku nasledujúcimi nekonečne mnohými spôsobmi:

$$\frac{1}{2}, \frac{2}{4}, \frac{3}{6}, \frac{4}{8}, \ldots.$$

Úloha 3.14 Ako sa dá v tvare zlomku nekonečne veľa spôsobmi zapísať racionálne číslo $\frac{3}{7}$?

Ale naším cieľom je mať na tomto liste papiera zapísané každé kladné racionálne číslo práve raz. Preto vezmeme vždy len zlomky p/q, ktoré sa už nedajú krátiť[11]. Číslo 1 je takto reprezentované zlomkom $\frac{1}{1}$, jedna polovica je reprezentovaná zlomkom $\frac{1}{2}$, pretože všetky iné zlomkové reprezentácie týchto čísiel sa dajú krátiť. Na našom nekonečnom liste papiera vygumujme všetky zlomky, ktoré sa dajú krátiť. Niektoré priesečníky riadkov a stĺpcov ostanú prázdne (obr. 3.21), ale to nám nebude prekážať.

[10] V podstate všetky.

[11] Najväčší spoločný deliteľ celých čísel p a q je 1.

	1	2	3	4	5	6	...
1	$\frac{1}{1}$	$\frac{1}{2}$	$\frac{1}{3}$	$\frac{1}{4}$	$\frac{1}{5}$	$\frac{1}{6}$	...
2	$\frac{2}{1}$		$\frac{2}{3}$		$\frac{2}{5}$		...
3	$\frac{3}{1}$	$\frac{3}{2}$		$\frac{3}{4}$	$\frac{3}{5}$		...
4	$\frac{4}{1}$		$\frac{4}{3}$		$\frac{4}{5}$		...
5	$\frac{5}{1}$	$\frac{5}{2}$	$\frac{5}{3}$	$\frac{5}{4}$		$\frac{5}{6}$	...
6	$\frac{6}{1}$				$\frac{6}{5}$		...
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋱

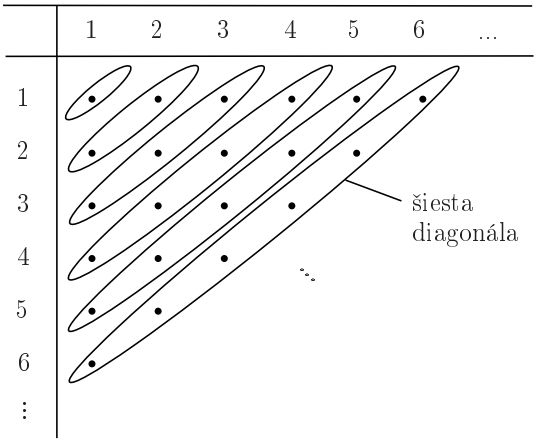
obr. 3.21

Všetky zlomky na obrázku 3.21 chceme očíslovať ako prvý, druhý, tretí atď. Je očividné, že očíslovanie prvkov na liste papiera nemôžeme dosiahnuť tak, že by sme očíslovali najprv čísla v prvom riadku, potom v druhom riadku, treťom riadku, atď. Dôvod, prečo to nejde je, že v prvom riadku je nekonečne veľa prvkov (zlomkov). Pokus očíslovať prvky na papieri týmto spôsobom zlyhá, pretože pri tomto postupe nikdy nezačneme číslovať prvky v druhom riadku. Prvý riadok jednoducho spotrebuje na svoje očíslovanie všetky čísla z $\mathbb{N}$. Z podobného dôvodu sa nedajú prvky nekonečného listu papiera očíslovať stĺpec po stĺpci. Ako máme teda postupovať? Očíslujeme čísla na našom liste papiera diagonálu po diagonále. ***k*-ta diagonála** nekonečného listu papiera obsahuje všetky pozície (obr. 3.22), pre ktoré platí, že súčet čísla riadka i a čísla stĺpca j dáva číslo $k+1$ $(i+j=k+1)$.

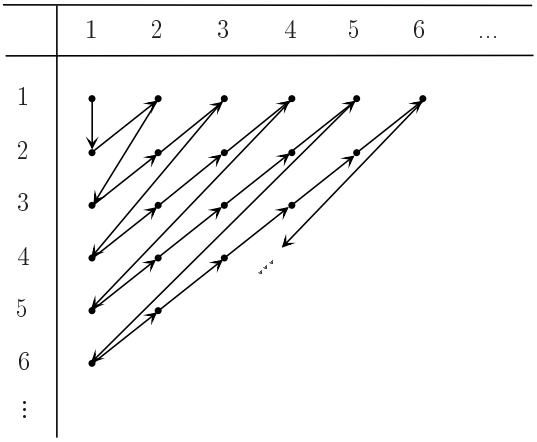
Takto prvá diagonála obsahuje len jeden prvok $\frac{1}{1}$. Druhá diagonála obsahuje dva prvky $\frac{2}{1}$ a $\frac{1}{2}$, tretia obsahuje dva prvky $\frac{3}{1}$ a $\frac{1}{3}$ a štvrtá diagonála obsahuje zlomky $\frac{4}{1}, \frac{3}{2}, \frac{2}{3}, \frac{1}{4}$. Vo všeobecnosti obsahuje k-ta diagonála presne k pozícií, teda nanajvýš k zlomkov.

Teraz očíslujeme pozície nekonečného listu papiera, a tým aj všetky kladné zlomky v poradí znázornenom na obrázku 3.23.

Pri tomto očíslovaní usporiadame diagonály zhora nadol a v rámci danej diagonály postupujeme zľava doprava. Podľa tejto stratégie číslovania a pozície zlomkov na obrázku 3.21, dostávame nasledujúce očíslovanie



obr. 3.22



obr. 3.23

kladných racionálnych čísel:

$$\frac{1}{1}, \frac{2}{1}, \frac{1}{2}, \frac{3}{1}, \frac{1}{3}, \frac{4}{1}, \frac{3}{2}, \frac{2}{3}, \frac{1}{4}, \frac{5}{1}, \frac{1}{5}, \frac{6}{1}, \frac{5}{2}, \frac{4}{3}, \frac{3}{4}, \frac{2}{5}, \frac{1}{6}, \ldots$$

Vzhľadom na dohodnuté označovanie je napríklad 1/1 nulté racionálne číslo, 2/1 je prvé racionálne číslo, 3 je tretie racionálne číslo, 1/3 je štvrté a 5/2 je dvanáste racionálne číslo.

Úloha 3.15 Rozšírte konečnú časť nekonečnej matice z obrázku 3.21 o ďalšie dva riadky a ďalšie dva stĺpce. Zistite, ktoré zlomky v našom očíslovaní racionálnych čísel budú mať poradové čísla $17, 18, 19, \ldots, 26, 27$.

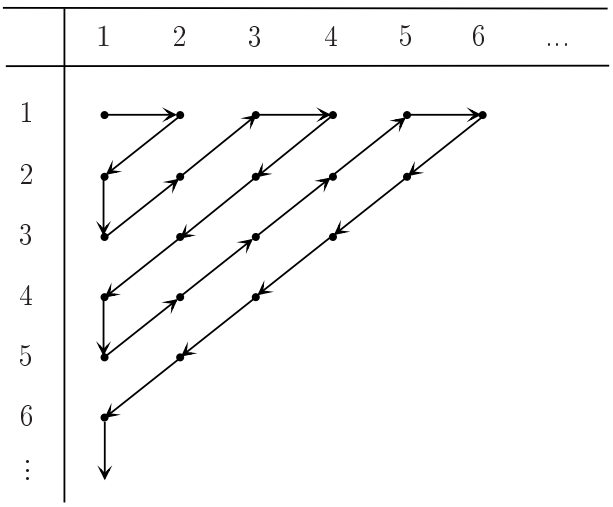
Pre zdôvodnenie korektnosti nášho číslovania je najdôležitejšie všimnúť si, že každý zlomok (každé racionálne číslo) dostal svoje poradové číslo. Samotný dôkaz tejto skutočnosti je jednoduchý. Nech p/q je ľubovoľný zlomok, ktorý sa nedá krátiť. Tento zlomok sa nachádza na políčku, kde sa pretína p-ty riadok a q-ty stĺpec, teda leží na diagonále $p+q-1$. Pretože každá diagonála obsahuje konečný počet pozícií, podarí sa nám postupne očíslovať prvky diagonál $1, 2, 3, \ldots, p+q-1$, takže dôjde aj k očíslovaniu prvkov diagonály $p+q-1$. Preto p/q tiež získa poradové číslo. Pretože i-ta diagonála obsahuje nanajvýš i racionálnych čísel, poradové číslo zlomku p/q je nanajvýš

$$1 + 2 + 3 + 4 + \ldots + (p + q - 1).$$

Z toho plynie platnosť rovnosti

$$|\mathbb{Q}^+| = |\mathbb{N}|.$$

Úloha 3.16 Obrázok 3.24 ukazuje iné očíslovanie zlomkov, ktoré je ale tiež založené na postupnom očíslovaní prvkov diagonál. Napíšte prvých 20 racionálnych čísiel podľa tohto očíslovania. Aké poradové číslo dostane racionálne číslo 7/3? Aké poradové číslo má racionálne číslo 7/3 v našom číslovaní znázornenom na obrázku 3.23?



obr. 3.24

Úloha 3.17 Hilbertov hotel je úplne vyťudnený, ako sa to niekedy mimo sezóny stáva. Naraz vypukne sezóna a príde nekonečne veľa nekonečných autobusov.

Autobusy sú očíslované v poradí

$$B_0, B_1, B_2, B_3, \ldots,$$

a je ich teda rovnako veľa ako $|\mathbb{N}|$. Pre každé i je v autobuse B_i nekonečne veľa miest

$$B_i(0), B_i(1), B_i(2), B_1(3), \ldots$$

a na každom mieste sedí práve jeden cestujúci. Ako recepčný dokáže ubytovať všetkých pasažierov zo všetkých týchto autobusov?

Úloha 3.18 (tvrdý oriešok) Dokážte rovnosť $|\mathbb{Q}| = |\mathbb{N}|$.

Úloha 3.19 (tvrdý oriešok) Definujme:

$$\mathbb{N}^3 = \{(i, j, k) \mid i, j, k \in \mathbb{N}\},$$

ako množinu všetkých trojíc (i, j, k), kde prvky trojíc i, j a k sú prirodzené čísla. Na každej pozícii tejto trojice sa môže nachádzať ľubovoľné číslo z nekonečnej množiny $\mathbb{N}$. Mohli by sme povedať, že $|\mathbb{N}^3| = \infty \cdot \infty \cdot \infty = \infty^3$. Ukážte, že platí $|\mathbb{N}^3| = |\mathbb{N}|$, a teda aj $\infty = \infty^3$.

3.3 Existujú rôzne veľké nekonečná, alebo prečo je reálnych čísiel viac ako prirodzených?

V predchádzajúcej časti sme sa zoznámili s Cantorovou koncepciou na porovnávanie mohutnosti množín. S prekvapením sme zistili, že nekonečno sa odlišuje od konečných objektov tým, že obsahuje časti, ktoré sú rovnako veľké ako sám celok. Pri hľadaní nekonečna, ktoré by bolo väčšie ako $|\mathbb{N}|$, sme nemali úspech. Zistili sme, že dokonca platí $|\mathbb{Q}^+| = |\mathbb{N}|$, i keď racionálne čísla sa nachádzajú na reálnej osi nekonečne hustejšie ako prirodzené čísla. Skutočnosť je ešte prekvapujúcejšia. Dá sa ukázať, že pre každé celé číslo k platí, že

$$\underbrace{|\mathbb{N}| \cdot |\mathbb{N}| \cdot \ldots \cdot |\mathbb{N}|}_{k \text{ krát}} = \underbrace{\infty \cdot \infty \cdot \ldots \cdot \infty}_{k \text{ krát}} = \infty^k$$

je zase len $|\mathbb{N}| = \infty$.

Nezistíme nakoniec, že všetky nekonečné množiny sú rovnako veľké? Naším cieľom je ukázať pravý opak tým, že ukážeme

$$|\mathbb{R}^+| > |\mathbb{N}|.$$

Tento výsledok sme možno na začiatku považovali za možný, ale po absolvovaní časti 3.2 môžeme pochybovať o tom, či táto nerovnosť platí. Reálne čísla majú podobné vlastnosti ako racionálne čísla. Neexistuje najmenšie pozitívne reálne číslo a medzi ľubovoľnými dvoma rôznymi reálnymi číslami sa nachádza nekonečne veľa reálnych čísel. Pretože $|\mathbb{N}| = |\mathbb{Q}^+|$, znamenala by nerovnosť $|\mathbb{R}^+| > |\mathbb{N}|$ automaticky tiež platnosť nerovnosti

$$|\mathbb{R}^+| > |\mathbb{Q}^+|.$$

Nebolo by to prekvapujúce? V nasledujúcej kapitole pochopíme lepšie kľúčový rozdiel medzi množinami $\mathbb{R}$ a $\mathbb{Q}$. Ukážeme dokonca ešte silnejší výsledok. Označme pomocou $[0, 1]$ množinu všetkých reálnych čísiel medzi 0 a 1, vrátane nuly a jednotky. Ukážeme, že platí

$$|[0, 1]| \neq |\mathbb{N}|.$$

Ako sa dá ukázať nerovnosť mohutnosti dvoch množín? Na dôkaz rovnosti nám stačí nájsť vhodné párovanie. To nemusí byť vždy jednoduché, ale v istom zmysle to nie je až také ťažké, lebo je to konštruktívna úloha. Skonštruujete vhodné párovanie, a tým je jednoducho práca ukončená. Pre nerovnosť $|A| \neq |B|$ potrebujeme dokázať, že **neexistuje párovanie** medzi A a B. Problém je v tom, že potenciálne môže existovať nekonečne veľa stratégií na konštrukciu hľadaného párovania. Ako sa dá vylúčiť možnosť úspechu pri všetkých možných postupoch na konštrukciu vhodného párovania? Nemôžeme jednu po druhej preveriť nekonečne veľa stratégií. Keď chceme ukázať, že niečo neexistuje, hovoríme o **dôkazoch neexistencie**.

> *Zdôvodniť nemožnosť existencie nejakého objektu s istými vlastnosťami, alebo vylúčiť možnosť nejakého javu sú najťažšie úlohy, aké si v prírodných vedách môžeme zadať.*

Slovo „nemožné“ nemôžeme takmer vysloviť a ak už ho predsa použijeme, musíme veľmi starostlivo upresniť jeho význam. Jeden známy fyzik mi raz rozprával, že existuje možnosť spätne vytvoriť vajíčko v pôvodnom stave z volského oka upečeného na panvici. Táto možnosť je založená na možnosti „obrátiť chod času“ fyzikálnych procesov[12] a nechať ich bežať späť do pôvodného stavu. Fyzici vraj dokážu vypočítať pravdepodobnosť, s akou sa podarí zrealizovať spätný beh procesu od volského oka k vajíčku. Pravdepodobnosť úspechu takého pokusu je tak nízka, že jeho úspech možno pokladať temer za zázrak, ale podstatné je, že táto pravdepodobnosť je väčšia ako nula, a teda táto možnosť existuje. Je veľa vecí

[12] ako to formuluje kvantová mechanika

a situácií, ktoré sa nám na prvý pohľad môžu zdať nemožné a napriek tomu môžu nastať.

V matematike pracujeme v umelom svete abstraktných matematických objektov a vďaka tomu v matematike môžeme úspešne vytvárať dôkazy o neexistencii matematických objektov. To ale nič nemení na skutočnosti, že aj tu patria dôkazy neexistencie k najťažším úlohám z hľadiska korektnej argumentácie.

Pokúsme sa najprv ukázať, že nie je možné očíslovať reálne číslo z intervalu $[0,1]$ a teda, že platí $|[0,1]| \neq |\mathbb{N}|$. Ako sme už naznačili, na dôkaz neexistencie očíslovania čísiel v $[0,1]$ použijeme metódu nepriameho dokazovania. To znamená, že predpokladáme opak toho, čo chceme dokázať. Takže predpokladáme, že existuje očíslovanie reálnych čísiel z intervalu $[0,1]$ a pomocou tohto predpokladu sa snažíme dostať k sporu[13].

Ak existuje očíslovanie množiny $[0,1]$, môžeme všetky reálne čísla z $[0,1]$ zapísať do postupnosti znázornenej v tabuľke na obrázku 3.25.

	0	1	2	3	4	…	i	…
1	0.	$\boxed{a_{11}}$	a_{12}	a_{13}	a_{14}	…	a_{1i}	…
2	0.	a_{21}	$\boxed{a_{22}}$	a_{23}	a_{24}	…	a_{2i}	…
3	0.	a_{31}	a_{32}	$\boxed{a_{33}}$	a_{34}	…	a_{3i}	…
4	0.	a_{41}	a_{42}	a_{43}	$\boxed{a_{44}}$	…	a_{4i}	…
⋮	⋮	⋮	⋮	⋮	⋮	⋱		
i	0.	a_{i1}	a_{i2}	a_{i3}	a_{i4}	…	a_{ii}	…
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋱

obr. 3.25

To znamená, že číslo

$$0, a_{11}a_{12}a_{13}a_{14}\ldots$$

[13]V prípade potreby sa môžeme najprv vrátiť k opisu metódy nepriamej argumentácie v prvej kapitole. Schéma nepriameho dokazovania hovorí, že ak je možné z nejakého tvrdenia Z odvodiť postupnosťou implikácií nejaké očividne neplatné tvrdenie, tak tvrdenie Z neplatí, a teda platí opak tvrdenia Z. Opakom existencie zobrazenia medzi $[0,1]$ a $\mathbb{N}$ je neexistencia zobrazenia medzi $[0,1]$ a $\mathbb{N}$.

je prvým číslom v našom očíslovaní čísiel z $[0,1]$. Symboly $a_{11}, a_{12}, a_{13}, \ldots$ sú decimálne cifry. Takže a_{11} je prvá decimálna cifra za desatinnou čiarkou, a_{12} je druhá cifra, a_{13} je tretia cifra za desatinnou čiarkou, atď. Vo všeobecnosti je

$$0, a_{i1}a_{i2}a_{i3}a_{i4}\ldots$$

i-te reálne číslo z intervalu $[0,1]$ v poradí nášho číslovania. Naša tabuľka je nekonečná v dvoch rozmeroch. Počet riadkov je $|\mathbb{N}|$ a počet stĺpcov je tiež $|\mathbb{N}|$. Stĺpec j obsahuje j-te cifry za desatinnou čiarkou všetkých reálnych čísiel v našom očíslovaní. Počet stĺpcov musí byť tiež nekonečný, pretože väčšinu reálnych čísiel nie je možné zapísať pomocou konečného počtu cifier za desatinnou čiarkou. Napríklad na zapísanie zlomku

$$\frac{1}{3} = 0,\overline{3} = 0,33333\ldots$$

potrebujeme nekonečne veľa miest za desatinnou čiarkou i keď táto reprezentácia je periodická. Čísla ako $\sqrt{2}/2$ alebo $\pi/4$ nie sú periodické a na ich presný zápis v tomto tvare decimálneho zápisu sa nemôžeme vyhnúť použitiu nekonečného počtu desatinných miest v ich presnom zápise.

Aby sme zvýšili názornosť uvedenej tabuľky, predstavíme si hypotetickú konkrétnu tabuľku ako na obrázku 3.26, kde sú namiesto symbolov a_{ij} konkrétne decimálne cifry. V tejto hypotetickej tabuľke sú očíslované reálne čísla z intervalu $[0,1]$.

Na prvom mieste je číslo $0,732110\ldots$, na druhom mieste $0,000000\ldots$, na treťom mieste $0,998103\ldots$, atď.

Teraz aplikujeme tzv. **diagonalizačnú metódu** na konštrukciu reálneho čísla z intervalu $[0,1]$, ktoré sa určite v tabuľke nenachádza (obr. 3.25). Táto skutočnosť je ale v spore s naším predpokladom, že v tabuľke máme očíslovanie reálnych čísiel z $[0,1]$, a teda nie je možné, aby tam nejaké číslo chýbalo. To znamená, že naša tabuľka neposkytuje očíslovanie reálnych čísiel z intervalu $[0,1]$ a teda, že neexistuje očíslovanie čísiel z $[0,1]$.

Naším cieľom je teda skonštruovať nejaké číslo c z intervalu $[0,1]$, ktoré sa určite nenachádza v nijakom riadku tabuľky na obr. 3.25. Číslo c budeme konštruovať postupne, jedno desatinné miesto po druhom. Číslo c zapíšeme ako

$$c = 0, c_1c_2c_3c_4\ldots c_i\ldots\ .$$

Vo všeobecnosti zvolíme decimálnu cifru $c_1 = a_{11} - 1$, ak $a_{11} \neq 0$ a položíme $c_1 = 1$, ak $a_{11} = 0$. Konkrétne na obrázku 3.26 to znamená, že by sme zvolili $c_1 = 6$, pretože $a_{11} = 7$. Vďaka tejto voľbe cifry c_1 vieme už

	0	1	2	3	4	5	6	...
1	0.	**7**	3	2	1	1	0	...
2	0.	0	**0**	0	0	0	0	...
3	0.	9	9	**8**	1	0	3	...
4	0.	2	3	4	**0**	7	8	...
5	0.	3	5	0	1	**1**	2	...
6	0.	3	1	4	0	5	**7**	...
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋱
i	0.	7	6	5	0	0	1	...
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋱

obr. 3.26

s istotou, že konštruované číslo c sa nenachádza v prvom riadku tabuľky na obr. 3.25 (obr. 3.26). Na druhé miesto za desatinnou čiarkou čísla c vezmeme cifru $c_2 = a_{22} - 1$ ak $a_{22} \neq 0$ a v prípade, keď $a_{22} = 0$ zvolíme $c_2 = 1$. Pre očíslovanie na obrázku 3.26 to znamená, že zvolíme $c_2 = 1$, pretože $a_{22} = 0$. Vďaka tejto voľbe sa číslo c s istotou odlišuje od čísla v druhom riadku tabuľky. Ďalej zvolíme c_3 tak, aby c_3 bolo rôzne od a_{33} a tým pádom konštruované číslo c neležalo v treťom riadku tabuľky (nebolo tretím číslom v hypotetickom očíslovaní).

Úplne všeobecne zvolíme $c_i = a_{ii} - 1$ pre $a_{ii} \neq 0$ a $c_i = 1$ pre $a_{ii} = 0$. Vďaka tejto voľbe neleží c v i-tom riadku tabuľky (c sa odlišuje od čísla v i-tom riadku tabuľky minimálne na i-tom desatinnom mieste). Týmto postupom získame po šiestich krokoch konštrukcie pre konkrétnu tabuľku z obrázku 3.26 číslo

$$0,617106\ldots.$$

Ľahko zistíme, že toto číslo je rôzne od všetkých čísel v prvých šiestich riadkoch tabuľky (obr. 3.26).

Vo všeobecnosti vidíme, že sa číslo c odlišuje od každého čísla v tabuľke (v našom hypotetickom očíslovaní) a to minimálne na i-tom desatinnom mieste od i-teho čísla tabuľky (očíslovania). To ale znamená, že tabuľka

na obrázku 3.25 nie je očíslovaním množiny $[0,1]$, pretože očíslovanie množiny $[0,1]$ znamená uviesť v poradí bezpodmienečne všetky prvky z $[0,1]$, a my vidíme, že c tam chýba. Takže bol náš predpoklad existencie očíslovania množiny $[0,1]$ chybný a na základe schémy nepriameho dôkazu môžeme uzavrieť naše úvahy tvrdením, že:

> *Neexistuje očíslovanie množiny* $[0,1]$*, a teda neexistuje ani párovanie medzi* $\mathbb{N}$ *a* $[0,1]$.

Úloha 3.20 Načrtnite začiatok tabuľky (podobne ako na obrázku 3.26) hypotetického očíslovania reálnych čísel z $[0,1]$, v ktorej na začiatku ležia čísla $1/4$, $1/8$, $\sqrt{2}/2$, 0, 1, $\pi/4$, $3/7$. K tejto tabuľke určte cifry $c_1, c_2, c_3, c_4, c_5, c_6$ a c_7 z c tak, aby bolo zaručené, že sa c odlišuje od všetkých týchto siedmych čísel.

Úloha 3.21 V hypotetickom očíslovaní leží v 100. riadku číslo $2/3$. Ktorú cifru čísla c a akým spôsobom zvolí v tomto prípade diagonalizačná metóda?

Úloha 3.22 Určte prvých 7 cifier čísla c za desatinnou čiarkou pre hypotetické očíslovanie množiny $[0,1]$ v tabuľke na obrázku 3.27.

	0	1	2	3	4	5	6	7	…
1	0.	**2**	0	0	1	7	8	0	…
2	0.	1	**7**	3	1	7	8	4	…
3	0.	1	6	**4**	3	3	3	3	…
4	0.	1	6	3	**0**	7	8	4	…
5	0.	1	6	3	1	**8**	8	4	…
6	0.	1	6	3	1	7	**9**	4	…
7	0.	1	6	3	1	7	8	**4**	…
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋱

obr. 3.27

Čo sme práve dokázali a aká bola naša argumentácia? Predpokladajme, že nám niekto povie, že má očíslovanie množiny $[0,1]$. Vyvinuli sme techniku (nazývanú diagonalizačná metóda), pomocou ktorej sme schopní každému jeho návrh očíslovania vrátiť s tým, že jeho očíslovanie nie je

úplné, pretože v ňom chýba aspoň jedno číslo z $[0,1]$. Vzhľadom na to, že sme to schopní urobiť pre každé hypotetické očíslovanie, neexistuje korektné očíslovanie prvkov množiny $[0,1]$.

Iný spôsob pohľadu na našu argumentáciu je založený na schéme nepriameho dôkazu, ktorú sme sa naučili v prvej kapitole. Naším cieľom bolo dokázať tvrdenie Z, že neexistuje očíslovanie reálnych čísiel z intervalu $[0,1]$. Naša schéma sa začína tvrdením $\overline{Z}$, čo je opakom Z a odvodí zo $\overline{Z}$ nejaký nezmysel (niečo nemožné). Vďaka tomu vieme podľa metódy nepriameho dôkazu, že $\overline{Z}$ nemôže platiť a musí platiť Z. Takto dosiahneme náš cieľ, dokázať platnosť Z. Tvrdenie $\overline{Z}$ (opak Z) je, že existuje očíslovanie množiny $[0,1]$. Predpokladajúc platnosť $\overline{Z}$ dospejeme k záveru, že v tomto hypotetickom očíslovaní prvkov množiny $[0,1]$ jeden prvok (číslo) c chýba. To je ale nezmysel, pretože v očíslovaní množiny $[0,1]$ nemôže chýbať nijaký prvok z $[0,1]$ a z tohto dôvodu neexistuje očíslovanie množiny $[0,1]$.

Vzhľadom na to, že nie je možné očíslovať čísla z intervalu $[0,1]$ (nie je možné popárovať ich s prirodzenými číslami z $\mathbb{N}$), nemôžeme očíslovať ani prvky z $\mathbb{R}^+$.

Úloha 3.23 Odôvodnite vlastnými slovami, prečo z neexistencie očíslovania množiny $[0,1]$ vyplýva tiež neexistencia očíslovania množiny $\mathbb{R}^+$.

[Rada: Môžete sa pokúsiť vysvetliť, ako by sa z každého očíslovania množiny $\mathbb{R}^+$ dalo získať očíslovanie množiny $[0,1]$. Prečo je takáto argumentácia korektná?].

Pretože $\mathbb{N} \subset \mathbb{R}^+$ a neexistuje párovanie medzi $\mathbb{N}$ a $\mathbb{R}^+$, smieme tvrdiť, že platí:

$$|\mathbb{N}| < |\mathbb{R}^+|.$$

Teda existujú dve rôzne veľké nekonečná a to $|\mathbb{N}|$ a $|\mathbb{R}^+|$. Možno dokonca ukázať, že existuje nekonečne veľa rôzne veľkých nekonečien. Tu ale upustíme od prezentácie a dokazovania týchto technických výsledkov, pretože ich nepotrebujeme na dosiahnutie nášho hlavného cieľa. Naším hlavným cieľom je v nasledujúcej kapitole ukázať, že existuje viac výpočtových problémov (úloh) ako algoritmov, a preto existujú úlohy, ktoré sa algoritmicky a teda automaticky pomocou počítačov nedajú riešiť.

Úloha 3.24 Pozmeňme trocha metódu diagonalizácie vzhľadom na tabuľku prezentovanú na obr. 3.25. Zvoľme $c_i = a_{i,2i} - 1$ ak $a_{i,2i} \neq 0$ a zvoľme $c_i = 1$ ak $a_{i,2i} = 0$.

a) Môžeme potom znova tvrdiť, že číslo $c = 0, c_1 c_2 c_3 c_4 \ldots$ sa v tabuľke nenachádza? Odôvodnite Vaše tvrdenie.

b) Zarámčekujte prvky $a_{i,2i}$ tabuľky na obrázku 3.25!
c) Aké budú cifry c_1, c_2 a c_3 pri tejto stratégii, ak sa budeme zamýšľať nad konkrétnou tabuľkou na obr. 3.27? Zdôvodnite, prečo sa takto vygenerované číslo $c = 0, c_1c_2c_3\ldots$ určite nenachádza v prvých troch riadkoch tabuľky (obr. 3.27).

3.4 Najdôležitejšie myšlienky v krátkosti

Dve nekonečná môžeme porovnávať, ak ich reprezentujeme pomocou množín. Na tomto základe Cantor postavil svoj koncept na porovnávanie veľkosti (mohutnosti) množín. Pritom využil nasledujúcu bačovskú metódu. Dve množiny sú rovnako veľké, ak môžeme prvky oboch množín spárovať. Nejaká nekonečná množina A je rovnako veľká ako $\mathbb{N}$, ak je možné očíslovať prvky A a tak ich usporiadať ako prvý, druhý, tretí, atď. Očíslovanie množiny A nie je nič iné ako párovanie A a $\mathbb{N}$. Na naše prekvapenie môžeme párovať $\mathbb{N}$ a $\mathbb{Z}$ i keď $\mathbb{N}$ je vlastná podmnožina $\mathbb{Z}$. Na základe tohto a podobných výsledkov sme spoznali, že vlastnosť

obsahovať vlastnú časť, ktorá je rovnako veľká ako celok,

je presne tou vlastnosťou, ktorá odlišuje konečné od nekonečného. Konečné množiny nemôžu mať túto vlastnosť. Pre nekonečné množiny je nevyhnutné mať túto vlastnosť.
Napriek tomu, že medzi dvoma prirodzenými číslami i a $i+1$ leží vždy nekonečne veľa racionálnych čísiel, očíslovali sme šikovným spôsobom racionálne čísla (nie podľa veľkosti), a tým sme ukázali, že platí $|\mathbb{N}| = |\mathbb{Q}^+|$. Dôsledkom je, že prirodzených čísiel je rovnako veľa ako kladných racionálnych čísiel.

Ďalej sme pomocou nepriameho dôkazu ukázali, že neexistuje očíslovanie reálnych čísiel a teda, že platí:

$$|\mathbb{N}| < |\mathbb{R}^+|.$$

Dôsledkom je, že existujú minimálne dve rôzne veľké nekonečná. Cieľom nasledujúcej kapitoly je ukázať, že počet rôznych programov sa rovná $|\mathbb{N}|$, a že počet rôznych algoritmických úloh sa rovná $|\mathbb{R}^+|$. Dôsledkom bude existencia úloh, pre ktoré neexistujú algoritmy na ich riešenie.

Zatiaľ sme ešte neukázali nijaký div informatiky. Zato sme ale spoznali podstatu nekonečna a metódu na porovnávanie nekonečných veľkostí, a tým sme predstavili matematický div, ktorý je jedným z najfascinujúcejších drahokamov vedy. Drahokamy ležia len zriedkavo pohodené na

ulici. V typickom prípade je potrebné vynaložiť nemalé úsilie na ich získanie. Preto sme sa aj poriadne zapotili, kým sme boli schopní zvládnuť prácu s nekonečnom. Preto nás nesmie prekvapiť, že cesta k divom informatiky je tiež náročná. Ale trpezlivosť sa vypláca. Vydržte ešte dve kapitoly a potom zažijete divy informatiky jeden za druhým. Neočakávané a elegantné riešenia zdanlivo bezvýchodiskových situácií zvýšia tep každého priateľa vedy. Len s trpezlivosťou a usilovnosťou dosiahneme poznanie, ktoré má skutočnú hĺbku a veľkú cenu.

Vzorové riešenia k vybraným úlohám

Úloha 3.1 Pre množiny $A = \{2, 3, 4, 5\}$ a $B = \{2, 5, 7, 11\}$ existuje $4! = 24$ rôznych párovaní. Príklad párovania je

$$(2, 11), (3, 2), (4, 5), (5, 7),$$

alebo

$$(2, 11), (3, 7), (4, 5), (5, 2).$$

Postupnosť dvojíc $(2, 2), (4, 5), (5, 11), (2, 7)$ nie je párovanie množín A a B, pretože prvok 2 z A sa nachádza v dvoch pároch $(2, 2)$ a $(2, 7)$ a prvok 3 z A ostal voľný (nenachádza sa v nijakom páre).

Úloha 3.7 (tvrdý oriešok) Definujme vzťah $|A| < |B|$ pre dve nekonečné množiny A a B tak, že $|A| < |B|$ práve vtedy, ak existuje vlastná podmnožina C množiny B ($C \subset B$), ktorú možno popárovať s množinou A. Naším cieľom je ukázať, že pomocou tejto definície možno dospieť k nezmyselnému výsledku $|\mathbb{N}| < |\mathbb{N}|$. Tým ukážeme, že táto definícia relácie menší nie je vhodná na porovnávanie nekonečných množín.

Zvoľme si v tejto definícii $A = \mathbb{N}$ a $B = \mathbb{N}$. Ako C si zvolíme $\mathbb{Z}^+$. Očividne je $C = \mathbb{Z}^+$ vlastná podmnožina množiny $B = \mathbb{N}$ ($C \subset B$), pretože $\mathbb{Z}^+ \subset \mathbb{N}$. Párovanie medzi množinami $C = \mathbb{Z}^+$ a $B = \mathbb{N}$ sme už ukázali. Tým sú splnené všetky podmienky definície, a teda musí platiť $|A| < |B|$. Pretože sme zvolili $A = \mathbb{N} = B$, dostávame $|\mathbb{N}| < |\mathbb{N}|$.

Úloha 3.9 Párovanie medzi množinami $\mathbb{N}$ a $\mathbb{Z}$ môžeme nájsť pomocou nasledujúceho očíslovania prvkov zo $\mathbb{Z}$ (usporiadania do poradia nultý, prvý, druhý, tretí, ...):

$$0, 1, -1, 2, -2, 3, -3, 4, -4, \ldots, i, -i, \ldots.$$

Z tohto očíslovania prvkov zo $\mathbb{Z}$ dostaneme párovanie množín $\mathbb{N}$ a $\mathbb{Z}$ tak, že každému číslu množiny $\mathbb{Z}$ priradíme jeho poradie v očíslovaní. Začiatok nášho párovania vyzerá takto:

$$(0, 0), (1, 1), (2, -1), (3, 2), (4, -2), (5, 3), (6, -3), \ldots .$$

Vo všeobecnosti vytvárame dvojice:

$$(0,0), (2i-1, i) \text{ a } (2i, -i)$$

pre všetky kladné celé čísla i.

Úloha 3.11 (tvrdý oriešok) Recepčný sa prichystá na hostí takto: najprv rozdelí izby na nekonečne veľa skupín nekonečnej veľkosti. Vždy, keď príde nejaká skupina hostí (je jedno či konečná alebo nekonečná), použije pre nových hostí nasledujúcu ešte nepoužitú skupinu voľných izieb.

Recepčný má dobré matematické vzdelanie (čo je v Hilbertovom hoteli pre každého zamestnanca nevyhnutným predpokladom na prijatie do zamestnania) a vie, že existuje nekonečne veľa prvočísiel

$$2, 3, 5, 7, 11, 13, 17, 19, \ldots .$$

Nech p_i je i-te prvočíslo v tejto rastúcej postupnosti prvočísiel. Pomocou p_i určíme i-tu nekonečne veľkú skupinu izieb ako:

$$\text{Skupina}(i) = \{p_i, p_i^2, p_i^3, p_i^4, \ldots, (p_i)^j, \ldots\} .$$

Napríklad Skupina(2)= $\{3, 9, 27, 81, \ldots\}$. Recepčný vie tiež (vďaka znalosti základnej vety aritmetiky), že nijaké číslo z $\mathbb{N}$ sa nenachádza vo viac ako jednej skupine izieb. To znamená, že aj keď postupne príde nekonečne veľa skupín hostí, môže recepčný prideľovať izby bez toho, aby musel presťahovávať už ubytovaných hostí. Nezáleží na tom, či i-ta skupina hostí je konečná alebo nekonečná, skupina izieb Skupina(i) je pre nich zarezervovaná. Ak označíme hostí i-tej skupiny ako

$$G_{i,1}, G_{i,2}, G_{i,3}, \ldots, G_{i,j}, \ldots$$

tak hosť $G_{i,1}$ dostane izbu p_i, hosť $G_{i,2}$ izbu p_i^2 atď. Vo všeobecnosti dostane hosť $G_{i,j}$ izbu $(p_i)^j$.

Čitateľ mohol nájsť aj iné správne riešenie, ako sme uviedli.

Úloha 3.13 Postupnosť dvojíc

$$(0,0), (1,1), (2,4), (3,9), (4,16), \ldots, (i, i^2), \ldots$$

je párovaním medzi $\mathbb{N}$ a $\mathbb{N}_{quad}$. Očividne je každý prvok z $\mathbb{N}$ práve v jednom páre a analogicky sa vyskytuje každé číslo z $\mathbb{N}_{quad}$ práve jedenkrát ako druhý prvok nejakej dvojice.

Úloha 3.21 Decimálna reprezentácia čísla $2/3$ je

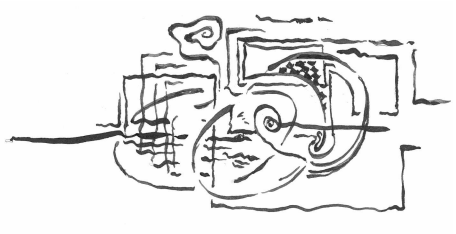
$$0, \overline{6} = 0, 666666 \ldots .$$

Takže na stom mieste za desatinnou čiarkou je cifra 6. Podľa metódy diagonalizácie zvolíme $c_{100} = 6 - 1 = 5$.

Úloha 3.22 Pre hypotetické očíslovanie množiny $[0, 1]$ z tabuľky obr. 3.27 dostaneme aplikovaním metódy diagonalizácie

$$c = 0,1631783\ldots\ .$$

Úloha 3.23 Úlohu vyriešime pomocou schémy nepriameho dôkazu z prvej kapitoly. Vieme, že neexistuje očíslovanie prvkov množiny $[0, 1]$. Našou úlohou je ukázať, že neexistuje očíslovanie prvkov z $\mathbb{R}^+$.
Predpokladajme opak nášho cieľa, t.j., že máme nejaké očíslovanie množiny $\mathbb{R}^+$. Zoberme toto očíslovanie ako postupnosť čísiel a škrtneme (vygumujeme) všetky čísla, ktoré nie sú z intervalu $[0, 1]$. To čo ostane, je postupnosť (očíslovanie) všetkých čísiel z $[0, 1]$. Už ale vieme, že neexistuje očíslovanie množiny $[0, 1]$, a teda musí platiť opak nášho predpokladu. Opak nášho predpokladu (že existuje očíslovanie $\mathbb{R}^+$), je náš cieľ (neexistuje očíslovanie množiny $\mathbb{R}^+$).



Diskutujte, mýľte sa,
robte chyby,
ale preboha myslite,
aj keď chybne, ale samostatne.

Gotthold Ephraim Lessing

Kapitola 4

Vypočítateľnosť alebo prečo existujú úlohy, ktoré sa nedajú vyriešiť na programovateľnom počítači

4.1 Ciele

V tretej kapitole sme objavili, že existujú rôzne veľké nekonečná. Napríklad, počet reálnych čísiel je väčší ako počet prirodzených čísiel. Nekonečná množina je presne rovnako veľká ako $\mathbb{N}$, keď jej prvky môžeme očíslovať ako prvý, druhý, tretí, atď.

Naším cieľom v tejto kapitole je ukázať, že existujú problémy (úlohy), ktoré nemožno riešiť pomocou algoritmov. Hlavná myšlienka je veľmi jednoduchá. Ukážeme, že existuje viac problémov ako je počet všetkých programov. To znamená, že musia existovať úlohy, ktoré sa nedajú riešiť algoritmicky a teda ani automaticky pomocou výpočtovej techniky. No neuspokojíme sa ale len s dôkazom, že existujú algoritmicky neriešiteľné úlohy. Mohli by sme si myslieť, že všetky algoritmicky neriešiteľné úlohy sú natoľko neprirodzené, že ich riešenie aj tak nikoho nezaujíma. Preto sa pousilujeme ukázať, že existujú zaujímavé praktické úlohy, ktoré

sa algoritmicky nedajú riešiť.

Táto kapitola je najťažšou v celej knihe, preto sa neznepokojujte ani nebuďte frustrovaní, ak sa vám nepodarí všetko pochopiť. Ani mnohí absolventi univerzitného štúdia informatiky celkom neovládajú túto matematicky náročnú časť štúdia. Úspechom je už len to, keď pochopíte prezentované objavy informatiky a ich význam. Pochopiť detailne celú cestu k týmto zásadným objavom informatiky si vyžaduje zvyčajne viacnásobné čítanie textu a premýšľanie o hlavných myšlienkach dôkazov. Je na vás, koľkokrát budete ochotní opakovane sa zaoberať touto témou.

Je dôležité vedieť, že každý môže čítať bez problémov všetky nasledujúce kapitoly aj v prípade, že nepochopil dôkazy v tejto kapitole.

4.2 Koľko rôznych programov možno napísať?

Koľko programov existuje? Prvá jednoduchá odpoveď je: „Nekonečne veľa“. Nepochybne pre každý program máme nejaký iný program, ktorý je o jeden riadok (jeden príkaz) dlhší, teda existuje dokonca nekonečne veľa možných dĺžok programov. Najviac nás ale zaujíma odpoveď na otázku, či je počet rôznych programov rovný $|\mathbb{N}|$. Ukážeme, že počet rôznych programov je rovnaký ako počet prirodzených čísel. Urobíme to tak, že očíslujeme všetky programy.

Začnime uvažovať o počte všetkých textov, ktoré možno napísať na počítači alebo písacom stroji. Pod textom rozumieme postupnosť **symbolov** používanej klávesnice. Všetky malé a veľké písmená latinskej abecedy chápeme ako symboly. Okrem nich množstvo ďalších symbolov ako

?, !, ·, $, /, +, *, atď.,

ktoré môžeme tiež používať. Na dôvažok máme ešte klávesnicu pre „prázdny symbol“, pomocou ktorej vkladáme medzeru medzi dve slová alebo dve vety. Aby sme naznačili výskyt medzery ako prázdneho symbolu, používame namiesto medzery symbol ␣. Dôležité je si uvedomiť, že medzera má v texte istý význam, a preto je rozumné s ňou narábať ako so symbolom. Takže za texty pokladáme nielen slová ako

„Janko“ a „mama“

s jasným významom, ale i nezmyselné postupnosti symbolov ako:

xyz*-+?!abe/

Od textu, v zmysle predchádzajúceho odseku, neočakávame nijaký význam. Je to jednoducho postupnosť symbolov, ktorá nemusí mať sémantickú interpretáciu. V informatike nazývame množinu použitých symbolov **abecedou** a hovoríme o **textoch nad abecedou**, ak sú texty zostavené zo symbolov tejto abecedy.

Pretože aj medzeru považujeme za symbol, môžeme obsah celej knihy[1] (napríklad tejto) považovať za text. Dopracovali sme sa k nasledovnému postrehu:

> *Každý text je konečný, ale texty môžu byť ľubovoľne dlhé, teda existuje nekonečne veľa rôznych textov.*

Pozrime sa teraz na podobnosť medzi textami a prirodzenými číslami. Každé prirodzené číslo možno reprezentovať konečným spôsobom pomocou symbolov (cifier) 0, 1, 2, 3, 4, 5, 6, 7, 8 a 9 v desiatkovej číselnej sústave. Veľkosť (dĺžka) reprezentácie rastie s hodnotou čísla, a preto môžu byť zápisy čísiel ľubovoľne dlhé[2].

Môžeme teda povedať:

> *„Počet všetkých textov nad abecedou počítačovej klávesnice sa rovná* $|\mathbb{N}|$*."*

Je to naozaj tak a zdôvodníme to pomocou očíslovania textov. Stačí ukázať, ako sa všetky texty dajú usporiadať do nekonečného telefónneho zoznamu. Nedá sa to ale spraviť rovnakým spôsobom, aký sa používa pri zostavení slovníkov. Podľa pravidiel usporiadania v slovníkoch najprv zoberieme do úvahy slová $a, aa, aaa, aaaa$, atď. a nikdy sa nedostaneme k očíslovaniu slov s inými písmenami ako a, pretože je nekonečne veľa textov, ktoré obsahujú len písmeno a. Preto musíme pri očíslovaní textov postupovať inak. Použijeme nasledujúce pravidlo:

> *Kratšie texty predchádzajú dlhším textom.*

To znamená, že v našej nekonečnej knihe sú na začiatku všetky texty dĺžky 1, potom všetky texty dĺžky 2, potom všetky texty dĺžky 3, atď. Ešte musíme určiť poradie pre texty rovnakej dĺžky. Ak by sme mali k dispozícii len písmená latinskej abecedy, mohli by sme texty rovnakej dĺžky usporiadať rovnako ako v slovníku. Teda najprv všetky texty začínajúce sa symbolom a, atď. Vzhľadom na to, že máme aj špeciálne symboly, ako

[1] okrem ilustrácií

[2] To znamená, že nemôžeme zhora ohraničiť dĺžku zápisu nejakou konštantou (pevným číslom). Ak by bola dĺžka najviac n, tak by sme mali k dispozícii nanajvýš 10^n rôznych reprezentácií čísiel, teda by sme mohli reprezentovať (zapísať) len konečne veľa čísiel.

?, !, *,+, atď., musíme sa najprv dohodnúť na poradí všetkých symbolov „abecedy klávesnice“. Poradie symbolov si môžeme zvoliť sami a nie je dôležité, ktoré si vyberieme. Potom

usporiadame texty rovnakej dĺžky rovnako ako v slovníku[3]

1	2	3	...	25	26	27	28	...	51	52	53	54	...	61	62
a	b	c	...	y	z	A	B	...	Y	Z	1	2	...	9	0

63	64	65	66	67	68	69	70	71	72	73	74	75	...	167
+	”	*	ç	&	!	.	:	,	;	?	$	?	...	␣

obr. 4.1

To znamená, že najprv sú texty, ktoré sa začínajú prvým symbolom v našom poradí. Ak sme napríklad usporiadali symboly abecedy klávesnice do poradia ako na obr. 4.2, začína naše očíslovanie textov

1 a
2 b
3 c
⋮
167 ␣
⋮

Texty dĺžky 5 sú usporiadané nasledujúco:

aaaaa
aaaab
aaaac
⋮
aaaa␣
aaaba
aaabb
aaabc
⋮

Prečo sme sa zapodievali textami?

„Každý program je text nad abecedou klávesnice.“

[3]Usporiadanie ako v slovníku zvyčajne voláme lexikografické.

Takže programy sú špeciálne texty, ktoré sú zrozumiteľné pre počítač. Počet programov nie je teda väčší ako počet textov, a preto môžeme tvrdiť:

> *„Počet programov sa rovná* $|\mathbb{N}|$*.“*

Ukázali sme, že počet programov je nekonečný a nie väčší ako $|\mathbb{N}|$. To, že $|\mathbb{N}|$ a počet programov sú rovnako veľké, vyplýva zo skutočnosti, že $|\mathbb{N}|$ je najmenšie nekonečno. Toto sme ale nedokázali. Keď chceme naše tvrdenie dokázať úplne čisto, bez použitia nedokázaného tvrdenia, potrebujeme vytvoriť párovanie medzi číslami a programami. Ako už vieme, očíslovanie je párovanie.

> *Programy očíslujeme tak, že z nekonečnej knihy všetkých textov vytvorených nad abecedou klávesnice, vymažeme tie, ktoré nepredstavujú program.*

Je dôležité všimnúť si, že vymazanie sa dá urobiť automaticky. Dajú sa napísať programy, ktoré pre zadaný vstupný text rozhodnú, či zodpovedá programu v nejakom uvažovanom programovacom jazyku. Takéto kontrolujúce programy nazývame **kompilátory**. Treba zdôrazniť, že

> *kompilátor skontroluje len, či je program syntakticky správny, ale nie, či je sémanticky správny.*

To znamená, že kompilátor skontroluje presne, či je text korektne zapísanou postupnosťou príkazov pre počítač, teda či je programom. Kompilátor nekontroluje, či je program algoritmom, teda, či program počíta niečo zmysluplné, alebo či sa nemôže stať, že by bežal v cykle do nekonečna.

V ďalšom si teda môžeme dovoliť uviesť nekonečný zoznam všetkých programov

$$P_0,\ P_1,\ P_2,\ P_3,\ \ldots,\ P_i,\ \ldots,$$

pričom $\boldsymbol{P_i}$ označuje ***i*-ty program**.

Prečo bolo také dôležité, ukázať, že počet programov, a tým súčasne aj algoritmov, nie je väčší ako $|\mathbb{N}|$? Odpoveď je, že počet všetkých možných problémov je väčší ako $|\mathbb{N}|$, teda existuje viac rôznych problémov ako programov. Z toho vyplýva, že existujú problémy, pre ktoré neexistujú algoritmy (metódy na ich riešenie).

Už v predchádzajúcej kapitole sme naznačili, že existuje príliš veľa problémov. Pre každé reálne číslo môžeme uvažovať nasledujúci problém.

Problem(c)

Vstup: prirodzené číslo n

Výstup: číslo c s presnosťou na n desatinných miest za desatinou bodkou

Hovoríme, že **algoritmus A_c rieši problém Problem(c)** alebo, že **A_c generuje číslo c**, keď pre ľubovoľné zadané číslo n vypočíta v desiatkovej sústave celú časť čísla c a n desatinných miest za desatinou bodkou.

Napríklad,

- pre $c = \frac{4}{3}$, musí $A_{\frac{4}{3}}$ pre vstup $n = 5$ vypočítať výsledok 1,33333.
- pre $c = \sqrt{2}$ musí $A_{\sqrt{2}}$ pre vstup $n = 4$ vypočítať číslo 1,4142.
- pre $c = \pi$ musí A_π pre vstup $n = 6$ vypočítať 3,141592.

Úloha 4.1 Čo je výstupom algoritmu $A_{\frac{17}{6}}$, ktorý pre vstup $n = 12$ generuje $\frac{17}{6}$? Čo sú výsledky algoritmu A_π, ktorý generuje π, pre vstupy $n = 2$, $n = 0$, $n = 7$ a $n = 9$?

Úloha 4.2 (tvrdý oriešok) Viete vymyslieť metódu na určenie (generovanie) ľubovoľného počtu desatinných miest čísla π? Vysvetlite ju!

V kapitole 3 sme dokázali, že počet reálnych čísiel je väčší ako $|\mathbb{N}|$, to znamená, že $|\mathbb{R}| > |\mathbb{N}|$. Pretože počet algoritmov nie je väčší ako $|\mathbb{N}|$, je viac reálnych čísiel ako algoritmov. Preto

> *existujú reálne čísla c, pre ktoré nie je* Problem(c) *algoritmicky riešiteľný.*

Dokázali sme, že sú reálne čísla, ktoré sa nedajú algoritmicky generovať. Rozumieme ale presne, prečo je to tak? Pokúsime sa vytvoriť si základy intuície, a tak demaskovať sivú eminenciu v pozadí. Prirodzené čísla, racionálne čísla, texty, programy, recepty a algoritmy majú dôležitú spoločnú vlastnosť.

> *„Všetky tieto objekty sa dajú reprezentovať konečným spôsobom.“*

Pre reálne čísla to ale neplatí. Keď je zápis reálneho čísla konečný, môžeme ho chápať ako text. Počet textov je ale menší ako počet reálnych čísiel, preto existujú reálne čísla, ktoré nemajú konečný zápis.

Čo to presne znamená? Konštruktívne opísať reálne číslo m znamená, že na základe opisu sme schopní získať kompletné číslo m, cifru po cifre.

Aj v prípade, keď má číslo m za desatinnou bodkou nekonečne veľa miest, na základe opisu vieme jednoznačne určiť cifru na ľubovoľnom desatinnom mieste čísla m. V tomto zmysle je konečný opis čísla m úplný. Teda na základe konečného opisu m máme algoritmus na generovanie tohto čísla. Napríklad $\sqrt{2}$ je konečná reprezentácia iracionálneho čísla $m = \sqrt{2}$ a vieme ho vypočítať s ľubovoľnou presnosťou, ktorú si určíme. Preto platí:

> *Reálne čísla, ktoré sa dajú reprezentovať konečným spôsobom, sú presne tie reálne čísla, ktoré sa dajú algoritmicky generovať. Máme aj reálne čísla, ktoré sa nedajú konečným spôsobom reprezentovať, a preto sa nedajú ani algoritmicky generovať.*

Úloha 4.3 Čo si myslíte? Čoho je viac? Reálnych čísiel s konečnou reprezentáciu alebo reálnych čísiel, ktoré nemajú konečnú reprezentáciu. Zdôvodnite vaše tvrdenie!

Teraz vidíme, že sú úlohy, na ktorých riešenie neexistuje algoritmus. S týmto poznaním sa ale neuspokojíme. Koho zaujíma úloha generovať číslo m, ktorého reprezentácia aj tak nie je konečná? Ako vôbec formulovať takúto úlohu konečným spôsobom? A okrem toho, ak sú toto jediné úlohy, ktoré nie sú algoritmicky riešiteľné, tak môžeme spokojne zabudnúť na celú takúto „umelú“ teóriu a venovať sa úlohám, ktoré sa vyskytujú v praxi. Takže čitateľovi musí byť zrejmé, že s takýmto poznaním sa nemôžeme uspokojiť. Musíme pokračovať v našom skúmaní, aby sme zistili, či existujú aj zaujímavé úlohy, ktoré sa dajú sformulovať konečným spôsobom a ktoré sa algoritmicky nedajú riešiť.

4.3 ÁNO alebo NIE, to je otázka alebo metóda diagonalizácie trochu inak

Najjednoduchšie problémy, ktorými sa zaoberá informatika, sú takzvané rozhodovacie problémy. Rozhodovací problém je o rozhodnutí, či daný objekt (alebo dané objekty) má (majú) určitú skúmanú vlastnosť. Napríklad dostaneme digitálnu fotografiu a máme rozhodnúť, či sa na nej nachádza stolička. Alebo, či je na fotografii človek, alebo ešte konkrétnejšie, či je to Aurel Stodola. Odpoveď musí byť jednoznačné „ÁNO“ alebo „NIE“. Iné odpovede nie sú povolené. Prirodzene, očakávame, že odpoveď bude správna.

V ďalšom budeme uvažovať o veľmi jednoduchých rozhodovacích problémoch. Nech je M ľubovoľná podmnožina $\mathbb{N}$. Je to teda množina obsahujúca prirodzené čísla. Špecifikujme **rozhodovací problém $(\mathbb{N}, M)$** takto:

Vstup: prirodzené číslo n z $\mathbb{N}$.

Výstup:

„ÁNO“ v prípade, že n je z M,
„NIE“ v prípade, že n nie je z M.

Ako M môžeme zobrať PRIM, kde

$$\text{PRIM} = \{2, 3, 5, 7, 11, 13, 17, 19, \ldots\}$$

je nekonečná množina všetkých prvočísiel. V takom prípade je (N, PRIM) rozhodovací problém, či dané číslo n je, alebo nie je prvočíslo. Problém ($\mathbb{N}$,$\mathbb{N}_{par}$) označuje rozhodovací problém, či dané číslo je párne alebo nepárne.

Pre každú podmnožinu M množiny $\mathbb{N}$, hovoríme, že **algoritmus A rozpozná množinu M** alebo, že **A rieši rozhodovací problém $(\mathbb{N}, M)$**, keď pre každý vstup n algoritmus A

(i) dá výsledok „ÁNO“, ak n patrí do M a

(ii) dá výsledok „NIE“, ak n nepatrí do M.

Niekedy používame namiesto „ÁNO“ cifru „1“ a namiesto „NIE“ cifru „0“. V prípade, že A pre vstup n odpovie „ÁNO“, hovoríme, že **algoritmus akceptuje číslo n**. Ak pre n odpovie „NIE“, hovoríme, že **algoritmus A zamietne číslo n**.

Keď pre rozhodovací problém $(\mathbb{N}, M)$ máme algoritmus, potom vravíme, že **$(\mathbb{N}, M)$ je algoritmicky riešiteľný** alebo, že

$(\mathbb{N}, M)$ je rozhodnuteľný.

Je očividné, že problém ($\mathbb{N}$, $\mathbb{N}_{par}$) je rozhodnuteľný; stačí preveriť, či je zadané prirodzené číslo párne alebo nepárne. Problém ($\mathbb{N}$, PRIM) je tiež rozhodnuteľný, lebo vieme, ako sa presvedčíme, či prirodzené číslo je prvočíslo a na základe tohto spôsobu nie je ťažké vytvoriť algoritmus.

Úloha 4.4 Najjednoduchší spôsob preverenia, či číslo n je prvočíslo, je vyskúšať vydeliť číslo n všetkými číslami od 2 po $n - 1$. V prípade, že nijaké z týchto čísiel nedelí n, je n prvočíslo. Takéto preverenie je veľmi nákladné. Ak chceme preveriť číslo 1000003, musíme vyskúšať miliónkrát deliteľnosť. Viete iný spôsob preverenia, pri ktorom je počet delení podstatne menší?

Úloha 4.5 (tvrdý oriešok) Vo formalizme z kapitoly 2 napíšte program, ktorý rozhodne problém (ℕ, QUAD), pričom

$$\text{QUAD} = \{1, 4, 9, 16, 25, \ldots\}$$

je množina štvorcov všetkých kladných čísiel (t.j. i^2, pre $i = 1, 2, 3, \ldots$).

Chceme ukázať, že existujú rozhodovacie problémy, pre ktoré neexistujú algoritmy. Takéto rozhodovacie problémy nazývame

nerozhodnuteľné alebo **algoritmicky neriešiteľné**.

Už sme sa dozvedeli, že môžeme vypísať všetky programy P_0, P_1, P_2, ... v pevnom poradí jeden za druhým. Neskôr sa dozvieme, že sa to dá urobiť dokonca algoritmicky. Vypísať algoritmicky algoritmy ale nie je také ľahké. Z tohto dôvodu začneme naše úsilie tým, že dokážeme dokonca niečo ťažšie. Ukážeme, že sú rozhodovacie problémy, ktoré sa nedajú vyriešiť nijakým programom. Čo to znamená presne? Kde je rozdiel medzi algoritmickou riešiteľnosťou a riešiteľnosťou pomocou programu?

Na pripomenutie: Každý algoritmus vieme prepísať ako program, ale nie každý program je algoritmus. Program môže vykonávať nezmyselnú činnosť a pre niektoré vstupy pracovať nekonečne dlho, zatiaľ čo algoritmus beží vždy len *konečný čas* a vypočíta *korektný* výsledok.

Nech je M podmnožina ℕ. Hovoríme, že **program P akceptuje množinu M**, keď pre každé zadané prirodzené číslo n

(i) P odpovie „ÁNO“, keď n patrí do M a

(ii) P odpovie „NIE“ alebo *pracuje nekonečne dlho*, keď n nepatrí do M.

V ďalšom texte bude označovať $M(P)$ množinu M, ktorú akceptuje P. Teda P môžeme chápať aj ako konečnú reprezentáciu potenciálne nekonečnej množiny $M(P)$.

Okamžite vidíme rozdiel medzi rozpoznaním množiny M algoritmom alebo jej akceptovaním programom. Pre vstupy z M musia oba pracovať korektne a v konečnom čase dodať správnu odpoveď „ÁNO“ (bod (i)). Pre čísla, ktoré nepatria do M, smie program na rozdiel od algoritmu pracovať nekonečne dlho bez toho, aby dal nejakú odpoveď. V tomto zmysle sú programy nadmnožina algoritmov[4]. Preto keď ukážeme, že pre množinu M neexistuje program, nebude pre M existovať ani algoritmus, a teda problém $(\mathbb{N}, M)$ je nerozhodnuteľný.

[4]Inak povedané, algoritmy sú špeciálne programy.

Aby sme skonštruovali takúto „ťažkú“ podmnožinu prirodzených čísiel, použijeme opäť diagonalizačnú metódu z kapitoly 3. Potrebujeme na to nasledujúcu reprezentáciu podmnožín prirodzených čísiel (obr. 4.2).

	0	1	2	3	4	...	i	$i+1$	...
M	0	1	0	0	1	...	1	0	...

obr. 4.2

M reprezentujeme ako nekonečnú postupnosť binárnych cifier. Postupnosť sa začína 0-tou pozíciou a na i-tom mieste je 1, keď i patrí do M. V prípade, že i nepatrí do M, napíšeme na i-te miesto v postupnosti 0. Množina M na obr. 4.2 teda obsahuje čísla 1, 4 a i. Prvky 0, 2, 3 a $i+1$ nepatria do M. Pre $\mathbb{N}_{par}$ vyzerá reprezentácia nasledujúco:

101010101010101010 ...

Pre PRIM je reprezentácia

0011010100010100 ...

Úloha 4.6 Určite prvých 17 miest v binárnej reprezentácii množiny QUAD.

Vytvorme teraz opäť dvojrozmernú tabuľku, ktorá je v oboch rozmeroch nekonečná. Stĺpce tabuľky označme postupnosťou

$$0, 1, 2, 3, 4, 5, \ldots, i, \ldots$$

všetkých prirodzených čísiel. Riadky označme postupnosťou

$$P_0,\ P_1,\ P_2,\ P_3,\ \ldots,\ P_i,\ \ldots$$

všetkých programov, ktoré prečítajú jediné číslo a ako odpoveď môžu vypísať len „ÁNO“ alebo „NIE“. Takéto programy sa dajú rozpoznať podľa toho, že obsahujú jediný príkaz „načítaj“ a príkazy výstupu smú vytlačiť len text „ÁNO“ alebo „NIE“. Každý takýto program P_i definuje jednoznačne množinu $M(P_i)$ všetkých prirodzených čísiel, pre ktoré program skončí s odpoveďou „ÁNO“. Všetky čísla, pre ktoré program neodpovie, alebo dá odpoveď „NIE“, nepatria do $M(P_i)$.

Teraz sú riadky tabuľky binárnymi zápismi množín $M(P_i)$. j-ty riadok (pozri obr. 4.3) obsahuje binárnu reprezentáciu množiny $M(P_j)$, ktorá je akceptovaná programom P_j. Políčko v i-tom riadku a j-tom stĺpci obsahuje jednotku, keď i-ty program akceptuje číslo j (pre vstup j skončí s „ÁNO“). Nula je na políčku v i-tom riadku a j-tom stĺpci, keď P_i pre vstup j odpovie „NIE“, alebo neodpovie vôbec.

Takto dostávame nekonečnú tabuľku, v ktorej riadkoch sú reprezentácie ***všetkých*** *podmnožín, ktoré môžu byť akceptované nejakým programom.*

	0	1	2	3	4	5	6	⋯	i	⋯	j	⋯
$M(P_0)$	**0**	1	1	0	0	1	0		1		0	
$M(P_1)$	0	**1**	0	0	0	1	1		0		0	
$M(P_2)$	1	1	**1**	0	0	1	0		1		1	
$M(P_3)$	1	0	1	**0**	1	0	1		1		0	
$M(P_4)$	0	0	0	1	**1**	0	1		0		1	
$M(P_5)$	1	1	1	1	1	**1**	1		1		1	
$M(P_6)$	1	0	1	0	0	0	**1**		0		1	
⋮												⋯
$M(P_i)$	0	1	1	0	0	1	0		**1**			
⋮												⋯
$M(P_j)$	1	0	1	0	1	1	1				**0**	
⋮												⋮

obr. 4.3

	0	1	2	3	4	5	6	⋯	i	⋯	j	⋯
DIAG	1	0	0	1	0	0	0		0		1	⋯

obr. 4.4

Teraz chceme ukázať, že je aspoň jedna podmnožina $\mathbb{N}$, ktorej nezodpovedá nijaký riadok v nekonečnej tabuľke (obr. 4.3). Ukážeme to tak, že skonštruujeme nekonečnú postupnosť DIAG núl a jednotiek, ktorá sa istotne v tabuľke nebude nachádzať. Konštrukciu DIAG a aj tomu zodpovedajúcej množiny $M(\text{DIAG})$ uskutočníme diagonalizačnou metódou.

Pozrieme sa na políčko a_{00}, kde sa pretína 0-tý riadok a 0-tý stĺpec. Keď $a_{00} = 0$ (obr. 4.3), to znamená, že 0 nepatrí do $M(P_0)$, dáme na 0-té miesto d_0 do DIAG 1 (takže do $M(\text{DIAG})$ zahrnieme 0). V prípade, že $a_{00} = 1$ (čiže ak 0 patrí do $M(P_0)$), dáme na 0-té miesto d_0 do DIAG hodnotu 0 (teda 0 nezahrnieme do $M(\text{DIAG})$). V prvom kroku sme určili iba prvý člen postupnosti DIAG a máme istotu, že DIAG sa líši od 0-tého riadku tabuľky (teda od $M(P_0)$) v aspoň v 0-tom prvku.

Rovnakým spôsobom postupujeme v druhom kroku. Pozrime sa na druhé políčko na uhlopriečke (diagonále) a_{11}, kde sa pretína prvý riadok s prvým stĺpcom. Naším cieľom je zvoliť hodnotu d_1 na prvej pozícii v DIAG tak,

aby sa $M(\text{DIAG})$ aspoň na tejto pozícii líšila od postupnosti $M(P_1)$. Preto položíme d_1 rovné 0 (1 nezaradíme do $M(\text{DIAG})$), keď $a_{11} = 1$ (1 patrí do $M(P_1)$). V prípade, že $a_{11} = 0$ (1 nepatrí do $M(P_1)$), položíme d_1 rovné 1 (1 zaradíme do $M(\text{DIAG})$).

Pre binárnu cifru a_{ij} z políčka kde sa pretína i-ty stĺpec a j-ty riadok predstavuje $\bar{a}_{ij}$ obrátenú hodnotu (obrátenie 1 je $\bar{1} = 0$ a obrátenie 0 je $\bar{0} = 1$). Na obr. 4.5 je znázornená situácia, ktorú sme dosiahli pri vytváraní DIAG po dvoch krokoch.

	0	1	2	3	4	$\cdots$	i	$i+1$	$\cdots$
DIAG	$\bar{a}_{00}$	$\bar{a}_{11}$	?	?	?	$\cdots$	?	?	$\cdots$

obr. 4.5

Prvé dva prvky v DIAG sú $\bar{a}_{00}$ a $\bar{a}_{11}$, takže $M(\text{DIAG})$ sa líši od $M(P_0)$ aj od $M(P_1)$. Zvyšné miesta v DIAG ešte nie sú určené a chceme ich určiť tak, aby sa DIAG líšil od každého riadku z tabuľky na obr. 4.3, a teda sa v nijakom z jej riadkov nenachádza.

Vo všeobecnosti zaručíme, aby sa DIAG líšil od i-teho riadku v tabuľke na obr. 4.3 nasledujúcim spôsobom. Keď políčko a_{ii}, nachádzajúce sa na priesečníku i-teho riadku a i-teho stĺpca, obsahuje 1 (i patrí do $M(P_i)$), potom položíme i-ty prvok d_i v DIAG rovný 0 (i nezaradíme do $M(\text{DIAG})$). V prípade, že $a_{ii} = 0$ (i nepatrí do $M(P_i)$), potom položíme $d_i = 1$ (i zaradíme do $M(\text{DIAG})$). Teda $M(\text{DIAG})$ sa líši od $M(P_i)$.

Takýmto spôsobom vytvoríme postupnosť DIAG tak, že sa nevyskytuje v nijakom z riadkov v tabuľke. Pre konkrétnu tabuľku na obr. 4.3 je na obr. 4.4 zodpovedajúca postupnosť DIAG. Vo všeobecnosti môžeme DIAG predstaviť ako na obr. 4.6.

	0	1	2	3	4	$\cdots$	i	$\cdots$
DIAG	$\bar{a}_{00}$	$\bar{a}_{11}$	$\bar{a}_{22}$	$\bar{a}_{33}$	$\bar{a}_{44}$	$\cdots$	$\bar{a}_{ii}$	$\cdots$

obr. 4.6

Takže platí, že

> $M(\text{DIAG})$ *neakceptuje nijaký program, a preto sa rozhodovací problém* $(\mathbb{N}, M(\text{DIAG}))$ *nedá rozhodnúť nijakým algoritmom.*

Definíciu M(DIAG) môžeme vyjadriť nasledujúcim stručným spôsobom:

$$M(\text{DIAG}) \quad = \quad \{n \in \mathbb{N} \mid n \text{ nepatrí do } M(P_n)\}$$

= množina všetkých prirodzených čísiel n takých,
že n nie je akceptované n-tým programom $M(P_n)$.

Úloha 4.7 Predstavte si, že prvých 10 riadkov a prvých 10 stĺpcov tabuľky všetkých programov vyzerá ako na obr. 4.7. Určte zodpovedajúcich prvých desať pozícií v DIAG.

	0	1	2	3	4	5	6	7	8	9	$\cdots$
$M(P_0)$	1	1	1	0	0	1	0	1	0	1	
$M(P_1)$	0	0	0	0	0	0	0	0	0	0	
$M(P_2)$	0	1	1	0	1	0	1	1	0	0	
$M(P_3)$	1	1	1	0	1	1	0	0	0	0	
$M(P_4)$	1	1	1	1	1	1	1	0	1	0	
$M(P_5)$	0	0	1	0	0	1	0	1	1	0	
$M(P_6)$	1	0	0	0	1	0	1	0	0	0	
$M(P_7)$	1	1	1	1	1	1	1	1	1	1	
$M(P_8)$	0	0	1	1	0	0	1	1	0	0	
$M(P_9)$	1	0	1	0	1	0	1	0	1	0	
$M(P_{10})$	0	0	1	0	0	0	1	1	0	1	
$\vdots$											$\ddots$

obr. 4.7

Úloha 4.8 (tvrdý oriešok) Skúmajme

$M(\text{2-DIAG})$ = množina všetkých párnych čísiel $2i$, takých, že $2i$ nepatrí do $M(P_i)$.

Je alebo nie je algoritmicky rozhodnuteľný problém $(\mathbb{N}, M(\text{2-DIAG}))$? Zdôvodnite vašu odpoveď. Nakreslite si k tomu aj obrázky analogické k obr. 4.3 a k obr. 4.4.

Úloha 4.9 (tvrdý oriešok) Pomôže vám riešenie predchádzajúcej úlohy (4.8) pri definovaní dvoch ďalších algoritmicky nerozpoznateľných podmnožín $\mathbb{N}$? Koľko algoritmicky neriešiteľných problémov sa dá vytvoriť metódou diagonalizácie?

Úloha 4.10 (tvrdý oriešok) Definujme

$M(\text{DIAG}_2)$ ako množinu všetkých párnych prirodzených čísiel $2i$, takých, že $2i$ nepatrí do $L(P_{2i})$.

Vieme povedať niečo o rozhodnuteľnosti $(\mathbb{N}, M(\text{DIAG}_2))$?

Už máme rozhodovací problém (ℕ, M(DIAG)), o ktorom vieme, že nie je algoritmicky riešiteľný. Ale ešte stále nie sme spokojní. Problém sa síce dá opísať konečným spôsobom (hoci najprv sme ho predstavili ako nekonečnú postupnosť), ale nie je to algoritmický návod na konštrukciu M(DIAG), lebo ako uvidíme neskôr, tabuľka na obr. 4.3 síce existuje, ale nedá sa algoritmicky (automaticky) vygenerovať. Okrem toho (ℕ, M(DIAG)) nezodpovedá nijakému prirodzenému praktickému problému.

4.4 Metóda redukcie alebo ako využiť úspešnú metódu riešenia problémov na dokazovanie neriešiteľnosti

Už vieme, že neriešiteľné problémy vieme opísať prostredníctvom diagonalizačnej metódy. Je to dobrá východisková pozícia. V tomto odseku pôjde o to, ako rozšíriť dôkazy algoritmickej neriešiteľnosti na ďalšie problémy. Hlavná myšlienka je zaviesť reláciu „**ľahší alebo rovnako ťažký**“ vzhľadom na algoritmickú riešiteľnosť.

Nech sú U_1 a U_2 dva problémy. Hovoríme:

U_1 **je ľahší, alebo rovnako ťažký ako** U_2**,**

alebo

U_2 **nie je ľahší ako** U_1

vzhľadom na algoritmickú riešiteľnosť a zapisujeme:

$$\boldsymbol{U_1 \leq_{Alg} U_2},$$

ak algoritmická riešiteľnosť U_2 zaručí algoritmickú riešiteľnosť U_1.

Čo to presne znamená? Keď máme

$$U_1 \leq_{Alg} U_2,$$

sú možné nasledujúce situácie:

- U_1 *a* U_2 *sú oba algoritmicky riešiteľné.*
- U_1 *je algoritmicky riešiteľný a* U_2 *je algoritmicky neriešiteľný.*
- U_1 *a* U_2 *sú oba algoritmicky neriešiteľné.*

Jediná situácia, ktorá je pre $U_1 \leq_{Alg} U_2$ vylúčená, je nasledujúca:

- *U_2 je algoritmicky riešiteľný a U_1 je algoritmicky neriešiteľný.*

Predstavte si, že ste dokázali sériu:

$$U_1 \leq_{Alg} U_2 \leq_{Alg} U_3 \leq_{Alg} \ldots \leq_{Alg} U_k$$

vzťahov medzi k problémami $U_1, U_2, \ldots, U_k$. Predpokladajme ďalej, že diagonalizačnou metódou sme schopní ukázať, že

U_1 je algoritmicky neriešiteľný.

Čo z toho vieme usúdiť? Pretože U_1 je najľahší zo všetkých problémov v sérii, sú ostatné problémy $U_2, U_3, \ldots, U_k$ v sérii aspoň také ťažké, ako U_1, a preto

sú problémy $U_2, U_3, \ldots, U_k$ algoritmicky neriešiteľné.

Presne takýmto spôsobom chceme viesť dôkaz algoritmickej neriešiteľnosti. Vďaka diagonalizačnej metóde máme už štartovací neriešiteľný problém U_1. Je to diagonalizačný problém $(\mathbb{N}, M(\mathrm{DIAG}))$. Otázkou ostáva, ako sa dá ukázať vzťah $U_1 \leq_{Alg} U_2$ medzi dvoma problémami?

Na tento účel použijeme metódu redukcie, vyvinutú v matematike na to, aby sa riešenie nových problémov dalo nájsť šikovne pomocou metód na riešenie iných, už vyriešených problémov. Metódu redukcie ilustrujeme na dvoch príkladoch.

Príklad 4.1 Predpokladajme, že máme metódu riešenia pre „normované“ kvadratické rovnice tvaru:

$$x^2 + px + q = 0,$$

to znamená, pre kvadratické rovnice s koeficientom 1 pri x^2. Metóda riešenia je daná prostredníctvom p-q-vzorca.

$$x_1 = -\frac{p}{2} + \sqrt{\left(\frac{p}{2}\right)^2 - q}$$

$$x_2 = -\frac{p}{2} - \sqrt{\left(\frac{p}{2}\right)^2 - q}.$$

Keď platí $\left(\frac{p}{2}\right)^2 - q < 0$, nemá rovnica riešenie v reálnych číslach.

Teraz chceme vyvinúť metódu na riešenie ľubovoľnej kvadratickej rovnice

$$ax^2 + bx + c = 0.$$

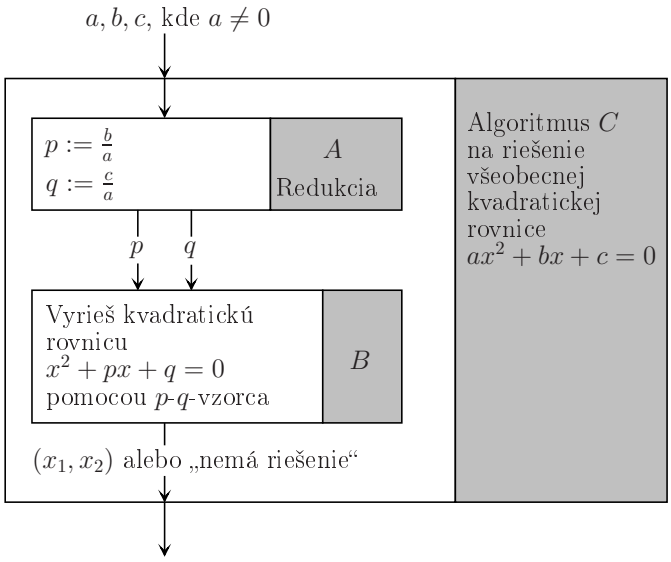
Namiesto odvodenia nového vzorca[5], *redukujeme* problém riešenia všeobecnej kvadratickej rovnice na riešenie normovanej kvadratickej rovnice.

Vieme, že riešenie hociakej rovnice sa nezmení, keď obe strany rovnice vynásobíme rovnakým číslom (rôznym od 0). Vynásobme teda obe strany kvadratickej rovnice číslom $\frac{1}{a}$.

$$\begin{aligned} ax^2 + bx + c &= 0 \qquad | \cdot \frac{1}{a} \\ a \cdot \frac{1}{a} \cdot x^2 + b \cdot \frac{1}{a} x + c \cdot \frac{1}{a} &= 0 \cdot \frac{1}{a} \\ x^2 + \frac{b}{a} x + \frac{c}{a} &= 0. \end{aligned}$$

Tým sme dostali normovanú kvadratickú rovnicu a tú vyriešime už danou známou metódou.

Algoritmicky je znázornená redukcia na obr. 4.8.



obr. 4.8

Časť A je algoritmus zodpovedajúci algoritmickej redukcii. Vypočítame v nej koeficienty p a q ekvivalentnej normovanej kvadratickej rovnice. Koeficienty p a q sú vstupom pre algoritmus B, ktorý rieši normovanú kvadratickú rovnicu tvaru $x^2 + px + q = 0$. B úlohu vyrieši. Výstup B sú

[5]Taký vzorec sme videli a programovali už v kapitole 2.

dve riešenia x_1 a x_2, alebo odpoveď „Nemá riešenie“, ktoré môžeme priamo prevziať ako výstup algoritmu C, ktorý rieši všeobecnú kvadratickú rovnicu. □

Úloha 4.11 Predpokladajme, že máme algoritmus B na riešenie lineárnych rovníc tvaru

$$ax + b = 0.$$

Vytvorte pomocou redukcie algoritmus na riešenie lineárnych rovníc tvaru

$$cx + d = nx + m,$$

kde c, d, n a m sú dané čísla a x je neznáma. Znázornite redukciu analogicky k schéme na obr. 4.8.

Redukciu z príkladu 4.1 nazývame **1-1-redukcia** (redukcia jedna k jednej). Je to najjednoduchšia možná redukcia, pri ktorej vstup pre problém U_1 (všeobecná kvadratická rovnica) priamo prerobíme na vstup pre problém U_2 (normovaná kvadratická rovnica) a výsledok algoritmu pre U_1 je „jedna k jednej“ prebraný výsledok pre U_2. Takto dostávame, že platí

$$U_1 \leq_{Alg} U_2. \tag{4.1}$$

Znamená to, že U_1 nie je algoritmicky ťažšie vyriešiť ako U_2, lebo algoritmus B pre U_2 môžeme prerobiť prostredníctvom redukcie na algoritmus C pre U_1 (obr. 4.8), a teda riešiteľnosť problému U_2 implikuje riešiteľnosť problému U_1.

V našom prípade si ešte môžeme všimnúť, že U_2 (riešenie normovanej kvadratickej rovnice) je špeciálnym prípadom U_1 (riešenie všeobecnej kvadratickej rovnice). Teda každý algoritmus pre U_1 je automaticky aj algoritmom pre U_2. To znamená, že platí:

$$U_2 \leq_{Alg} U_1. \tag{4.2}$$

Na základe uvedeného môžeme tvrdiť (z (4.1) a (4.2)), že U_1 a U_2 sú algoritmicky rovnako ťažké, čo znamená, že buď sú oba algoritmicky riešiteľné, alebo oba algoritmicky neriešiteľné. Prirodzene, v tomto špeciálnom prípade kvadratických rovníc platí prvá možnosť.

Nie vždy musia byť redukcie takéto jednoduché. Na to aby sme ukázali, že platí:

$$U_1 \leq_{Alg} U_2,$$

môže byť nevyhnutné, aby sme použili viackrát algoritmus B, ktorý rieši U_2, alebo aby sme ešte ďalej spracovávali výsledky B. Ilustrujeme to na príklade 4.2 a za ním v nasledujúcej úlohe.

Príklad 4.2 Všetci poznáme Pytagorovu vetu, ktorá hovorí, že v pravouhlom trojuholníku (obr. 4.9) platí:

$$c^2 = a^2 + b^2,$$

alebo presnejšie slovami:

> *Obsah štvorca nad preponou (najdlhšou stranou) pravouhlého trojuholníka sa rovná súčtu obsahov štvorcov nad oboma odvesnami (kratšími stranami).*

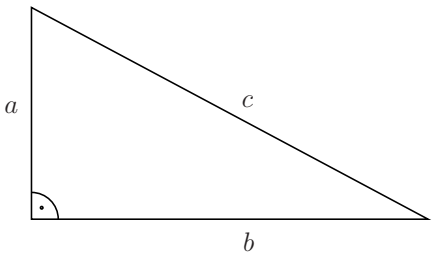
Teda máme algoritmus $B_\triangle$, ktorý pre zadané dve dĺžky strán pravouhlého trojuholníka vypočíta dĺžku tretej strany. Napríklad, keď poznáme a a b, vypočítame c takto:

$$c = \sqrt{a^2 + b^2}.$$

Keď poznáme a a c, vypočítame b nasledujúco:

$$b = \sqrt{c^2 - a^2}.$$

Označme $U_\triangle$ problém výpočtu veľkosti chýbajúcej strany pravouhlého trojuholníka.



obr. 4.9

Predstavme si teraz novú úlohu U_{Pl}. Daný je rovnostranný trojuholník (obr. 4.10) so stranami dĺžky m. Úlohou je vypočítať plochu tohto trojuholníka. Je očividné (obr. 4.10), že plocha trojuholníka je

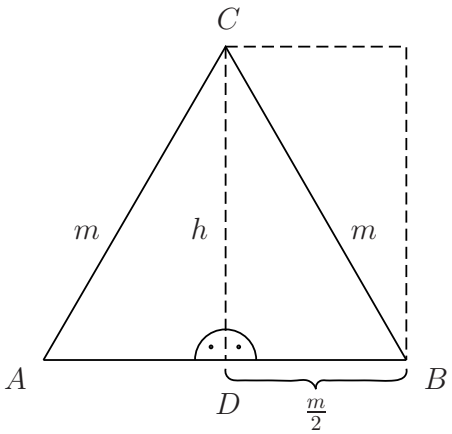
$$\frac{m}{2} \cdot h,$$

pričom h je výška trojuholníka (obr. 4.10).

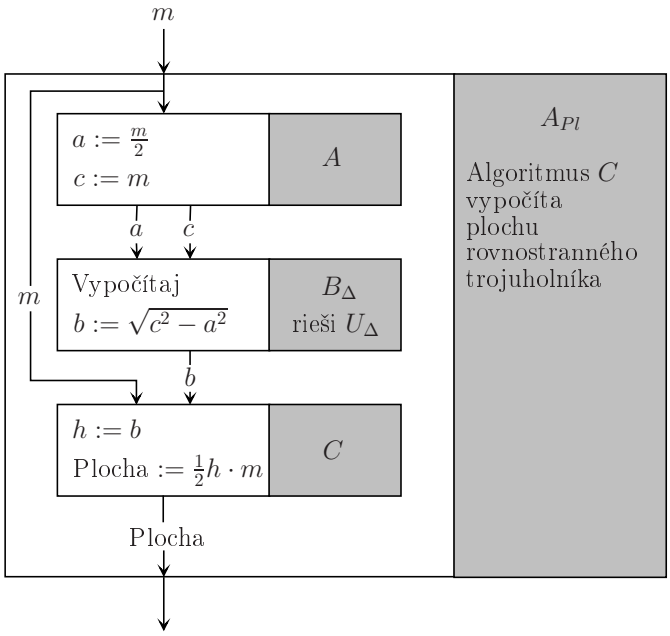
Vieme ukázať, že:

$$U_{Pl} \leq_{Alg} U_\triangle,$$

teda redukovať riešenie U_{Pl} na riešenie $U_\triangle$. Ako sa to dá urobiť, ukazujeme na obr. 4.11.



obr. 4.10

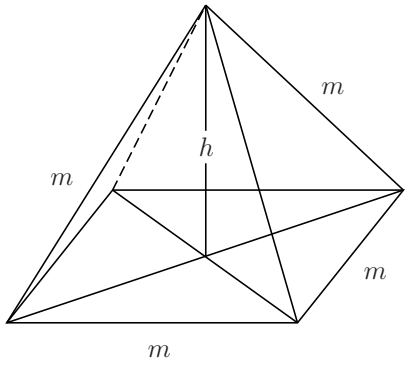


obr. 4.11

Na riešenie U_{Pl} vytvoríme algoritmus A_{Pl} za predpokladu, že máme algoritmus $B_\triangle$ na riešenie $U_\triangle$ (výpočet dĺžky chýbajúcej strany pravouhlého trojuholníka). Na obr. 4.11 vidíme, že na výpočet plochy potrebujeme výšku h trojuholníka. Veľkosť h je dĺžka strany CD pravouhlého trojuholníka DBC. Vidíme, že dĺžka a strany DB sa rovná $m/2$ a je zrejmé,

že dĺžka c prepony v trojuholníku DBC je m. Teda algoritmus A nastaví zodpovedajúco (obr. 4.11) hodnoty a a c. Potom použijeme algoritmus $B_\triangle$ pre $U_\triangle$, aby sme vypočítali chýbajúcu veľkosť $h = b$. Nakoniec vypočíta algoritmus C na základe hodnôt m a b plochu $\triangle ABC$. □

Úloha 4.12 Pozrime sa na úlohu U_{Pyr}, v ktorej vypočítame výšku h pyramídy so štvorcovou základňou veľkosti $m \times m$ a dĺžkami hrán m (obr. 4.12). Vyriešte



obr. 4.12

úlohu tak, že ukážete $U_{Pyr} \leq_{Alg} U_\triangle$. Nakreslite analogickú redukciu, ako je na obr. 4.11.

[Návod: Všimnite si, že pri redukcii musíte použiť dvakrát algoritmus $B_\triangle$ na riešenie $U_\triangle$.]

Úloha 4.13 (tvrdý oriešok) Nech je U_{2lin} problém riešenia systému dvoch lineárnych rovníc:

$$\begin{aligned} a_{11}x + a_{12}y &= b_1 \\ a_{21}x + a_{22}y &= b_2 \end{aligned}$$

s dvomi neznámymi x a y. Nech U_{3lin} je problém riešenia systému troch lineárnych rovníc:

$$\begin{aligned} a_{11}x + a_{12}y + a_{13}z &= b_1 \\ a_{21}x + a_{22}y + a_{23}z &= b_2 \\ a_{31}x + a_{32}y + a_{33}z &= b_3 \end{aligned}$$

s tromi neznámymi x, y a z. Ukážte, že $U_{3lin} \leq_{Alg} U_{2lin}$.

Videli sme, ako sa dajú vyvinúť prostredníctvom metódy redukcie určitých problémov metódy riešenia iných problémov. Takže redukcia slúži na šírenie algoritmickej riešiteľnosti.

My ale nechceme teraz používať redukciu ako pomôcku na vývoj algoritmov. Prostredníctvom redukcie chceme šíriť algoritmickú neriešiteľnosť (negatívne správy). Ako sa dá obrátiť zmysel používania metódy zo získavania pozitívnych výsledkov na získavanie negatívnych výsledkov? Naznačili sme to už na začiatku tejto kapitoly. Keď prostredníctvom redukcie vieme dokázať, že:

$$U_1 \leq_{Alg} U_2$$

a vieme, že U_1 je algoritmicky neriešiteľný, potom musí byť algoritmicky neriešiteľný aj U_2.

Je malý rozdiel v dôkaze

$$U_1 \leq_{Alg} U_2$$

za účelom šírenia algoritmickej riešiteľnosti alebo za účelom šírenia algoritmickej neriešiteľnosti. Pri pozitívnych výsledkoch sme mali nejaký algoritmus pre U_2 a niečo sme naprogramovali, aby sme dostali algoritmus pre U_1. Pri šírení negatívnych výsledkov o neriešiteľnosti, prirodzene nemáme nijaký algoritmus pre U_2. Iba *predpokladáme, že nejaký existuje.* A keď ho máme, potom s jeho pomocou vytvoríme algoritmus pre U_1. To znamená, že musíme pracovať s hypotetickou existenciou algoritmu A_2 pre U_2, aby sme mohli ako dôsledok usúdiť, že existuje algoritmus A_1 pre U_1.

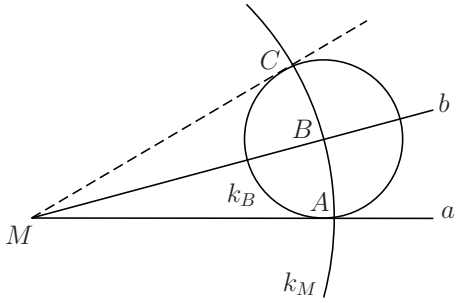
Použitie redukcie pre dôkaz algoritmickej riešiteľnosti zodpovedá priamemu dôkazu (priamej argumentácii), ktorú sme predstavili v kapitole 1. Použitie redukcie na dôkaz neexistencie algoritmu pre analyzovanú úlohu zodpovedá presne nepriamemu dôkazu, ako sme ho uviedli v kapitole 2.2. Aby sme sa opreli o niečo známe, uvedieme najprv príklad z matematiky a až potom prejdeme k algoritmike.

Príklad 4.3 Vieme, že sa nedá rozdeliť uhol na tri rovnako veľké časti len s pomocou kružidla a pravítka. Teda neexistuje metóda v tvare postupnosti jednoduchých krokov (jednoduchého použitia pravítka a kružidla), ktorou by sa dal ľubovoľný uhol geometricky rozdeliť na tri rovnako veľké časti. Dôkaz nie je zrejmý a ostaneme radšej pri tom, že toto tvrdenie matematikom uveríme.

Na druhej strane zo školy vieme, že pravítkom a kružidlom vieme uhol zdvojnásobiť alebo rozdeliť na rovnaké polovice.

Na obr. 4.13 ukážeme príklad, ako sa dá zdvojnásobiť uhol $\angle ab$, ktorý zvierajú priamky a a b. KP-algoritmus[6] na zdvojnásobenie uhla pracuje

[6] kružidlo-pravítko-algoritmus



obr. 4.13: Zdvojnásobenie uhla

nasledovne:

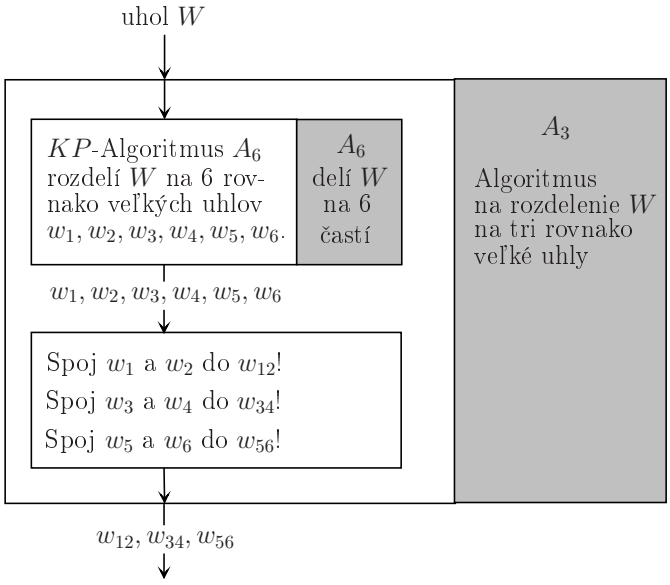
1. Zober do kružidla ľubovoľnú vzdialenosť r a nakresli kružnicu k_M so stredom M (priesečník a a b) a polomerom r.
2. Označ ako A priesečník kružnice k_M a priamky a, priesečník k_M a b označ B.
3. Do kružnice zober vzdialenosť $\overline{AB}$ medzi A a B a nakresli kružnicu k_B so stredom v bode B a polomerom $\overline{AB}$.
4. Označ ako C priesečník k_M a k_B, ktorý je rôzny od A.
5. Spoj pravítkom body M a C.

Teraz vieme, že uhol $\angle AMC$ medzi priamkou a a priamkou, ktorá vedie cez body M a C je dvakrát taký veľký, ako pôvodný uhol $\angle ab = \angle AMB$.

To bolo len na pripomenutie. Naša úloha je ukázať, že neexistuje KP-algoritmus, ktorý by ľubovoľný uhol rozdelil na šestiny (na šesť rovnako veľkých uhlov). Zdôvodniť niečo také pravdepodobne nebude o nič ľahšie, ako dokázať neexistenciu algoritmu na rozdelenie uhla na tretiny. Nemusíme ale ísť touto ťažkou cestou, keď vieme, že s pravítkom a kružidlom sa uhol nedá rozdeliť na tretiny. Túto skutočnosť môžeme využiť pri našom dôkaze.

Ako budeme postupovať? Zoberme si opak toho, čo chceme dokázať (vieme rozdeliť uhol na šesť rovnako veľkých uhlov) a ukážeme, že z takéhoto predpokladu vieme uhol rozdeliť aj na tretiny, čo ale odporuje už známej skutočnosti. Presnejšie povedané, predpokladáme, že máme KP-algoritmus A_6 na rozdelenie uhla na šestiny a pomocou A_6 vytvoríme KP-algoritmus A_3 na rozdelenie uhla na tretiny. Pretože A_3 neexistuje, usúdime, že ani A_6 nemôže existovať.

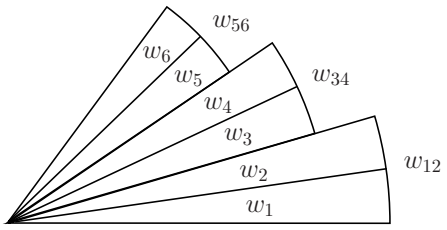
Opíšeme redukciu rozdelenia uhla na šestiny, na rozdelenie uhla na tretiny nasledujúco (obr. 4.14). Predpokladáme, že existuje KP-algoritmus A_6 na rozdelenie uhla na šestiny. Najprv použijeme A_6, aby sme daný uhol W rozdelili na šestiny. Dostaneme 6 rovnako veľkých uhlov w_1, w_2, w_3, w_4, w_5, w_6 (pozri obr. 4.15). Potom spojíme dva susedné uhly, počínajúc s w_1 a w_2 a dostaneme tri rovnako veľké uhly w_{12}, w_{34}, w_{56} (obr. 4.15). Rozdelenie W na tieto tri uhly zodpovedá jeho rozdeleniu na tretiny.



obr. 4.14

V reči nepriamej argumentácie (nepriameho dôkazu) hovorí redukcia zobrazená na obr. 4.14 o nasledujúcom dôsledku:

> *„Keď sa každý uhol dá KP-algoritmom rozdeliť na šestiny, potom sa dá KP-algoritmom každý uhol rozdeliť aj na tretiny.“*



obr. 4.15

Podľa definície implikácie platnosť dokázanej implikácie vylučuje druhú možnosť zo štyroch možných prípadov na obr. 4.16. Ak zoberieme do úva-

možnosť	rozdelenie na šestiny	rozdelenie na tretiny
1	dá sa	dá sa
2	dá sa	nedá sa
3	nedá sa	dá sa
4	nedá sa	nedá sa

obr. 4.16

hy ešte skutočnosť, že rozdelenie na tretiny sa nedá uskutočniť, musíme vylúčiť aj situácie č. 1 a 3, pre ktoré platí, že „rozdelenie na tretiny sa dá uskutočniť“. Jediná možná situácia, ktorá ostala, je č. 4 a tá obsahuje „rozdelenie na šestiny sa nedá uskutočniť“, a teda môžeme uzavrieť, že ľubovoľný uhol sa nedá rozdeliť nijakým KP-algoritmom na šestiny. □

Úloha 4.14 Problém delenia na tretiny si môžeme predstaviť aj v zjednodušenom tvare. Úlohou je pre ľubovoľný uhol W skonštruovať pravítkom a kružidlom uhol V tak, že W je trikrát väčší ako V. Dá sa dokázať, že aj takéto zjednodušenie formulácie nič nezmení na KP-neriešiteľnosti. Urobte podobný dôkaz aj s obrázkom, ako je obr. 4.14. Ukážte, že nijakým KP-algoritmom sa nedá skonštruovať uhol

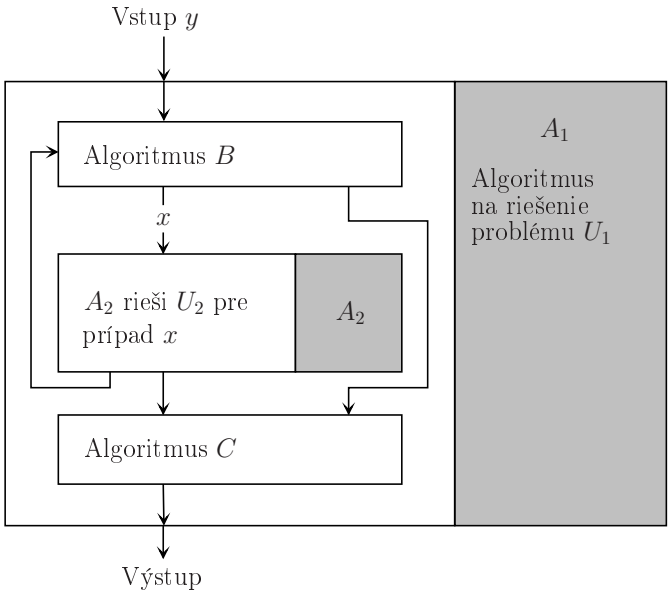
(i) veľkosti $\frac{1}{6}$,
(ii) veľkosti $\frac{1}{9}$,

ľubovoľného daného uhla.

Vráťme sa zo sveta KP-algoritmov naspäť do sveta všeobecných algoritmov. Náš problém diagonalizácie tu má podobné miesto ako delenie uhla na tretiny pri KP-algoritmoch. Z jeho algoritmickej neriešiteľnosti chceme usúdiť algoritmickú neriešiteľnosť ďalších problémov.

Schéma redukcie pre $U_1 \leq_{Alg} U_2$ je znázornená na obr. 4.17.

Algoritmus A_1 na riešenie U_1 je vytvorený nasledovne. Vstup pre U_1 najprv spracuje algoritmus B. Algoritmus B vie pre vstup generovať prípady x problému U_2 pre A_2. Predpokladáme, že A_2 vypočíta pre x správne riešenie. Ako vidíme na obr. 4.17, A_2 môžeme pritom použiť viackrát. Nakoniec spracuje algoritmus C všetky získané medzivýsledky a vypočíta definitívny výsledok pre prípad y problému U_1. Pritom je dôležité všimnúť si, že podprogramy A_2, B a C sú algoritmy, a preto výsledok vždy vypočítajú v konečnom čase. A_2 smie použiť B len konečne veľakrát,



obr. 4.17

preto sa aj cyklus obsahujúci B a A_2 vykoná tiež len konečne veľakrát. Takže môžeme usúdiť, že A_1 je tiež algoritmus, pretože pre každý vstup v konečnom čase korektne odpovie.

Teraz si predstavíme dva nové rozhodovacie problémy, ktoré majú význam pri vývoji a obzvlášť pri testovaní softvéru.

UNIV (univerzálny problém)
Vstup: program P a vstup $i \in \mathbb{N}$ pre P
Výstup: ÁNO, v prípade, že P akceptuje vstup i, to znamená, že i patrí do $M(P)$
NIE, v prípade, že P neakceptuje vstup i
(to znamená, že buď P skončí a zamietne vstup i, alebo P pracuje pre vstup i nekonečne dlho).

HALT (problém zastavenia)
Vstup: program P a prirodzené číslo i
Výstup: ÁNO, v prípade, že P sa zastaví pre vstup i
(to znamená, že P pracuje pre i konečne dlho).
NIE, v prípade, že P sa nezastaví pre vstup i
(to znamená, že P sa dostane pri práci so vstupom i do nekonečného cyklu).

Je zrejmé, že problém zastavenia je základnou otázkou pri testovaní softvéru. Vieme, že za algoritmy môžeme považovať len také programy, ktoré nebudú pracovať nekonečne dlho. Preto sa každý nový program, v rámci testovania jeho správnosti, preveruje, či je vždy (pre každý vstup) zaručené, že program v konečnom čase vypočíta výsledok. Problém zastavenia HALT je najjednoduchšou formou testovania. Pýtame sa len, či sa program P zastaví pre konkrétny vstup i (skutočná otázka je, či zastane pre všetky vstupy). Neskôr sa dozvieme, že už aj takto zjednodušená otázka sa algoritmicky nedá zodpovedať.

Univerzálny problém súvisí priamo s kontrolou správnosti programu P pre rozhodovací problém. Testujeme, či P pre vstup i odpovie správne ÁNO alebo NIE. Teraz by hocikto mohol povedať, to predsa vieme ľahko urobiť: Simulujme prácu P pre i a pozrime sa, či P odpovie ÁNO alebo NIE. To by sme skutočne mohli urobiť, keby sme mali zaručené, že P je algoritmus (to znamená, že P na svojom vstupe i zastane). To ale nemáme zaručené. Keď P bude na vstupe i pracovať nekonečne dlho, simulovali by sme prácu P na vstupe i nekonečne dlho a nedostaneme odpoveď na otázku, či P číslo i akceptuje alebo nie. My ale chceme navrhnúť algoritmus pre univerzálny problém, tento algoritmus nesmie počítať nekonečne dlho, teda sa nemôže realizovať nekonečná simulácia.

Uvedené úvahy ukazujú, že problém zastavenia a univerzálny problém sú vzájomne silne previazané. Naozaj, ukážeme, že sú rovnako ťažké.

Najprv ukážeme, že platí:

$$\text{UNIV} \leq_{Alg} \text{HALT},$$

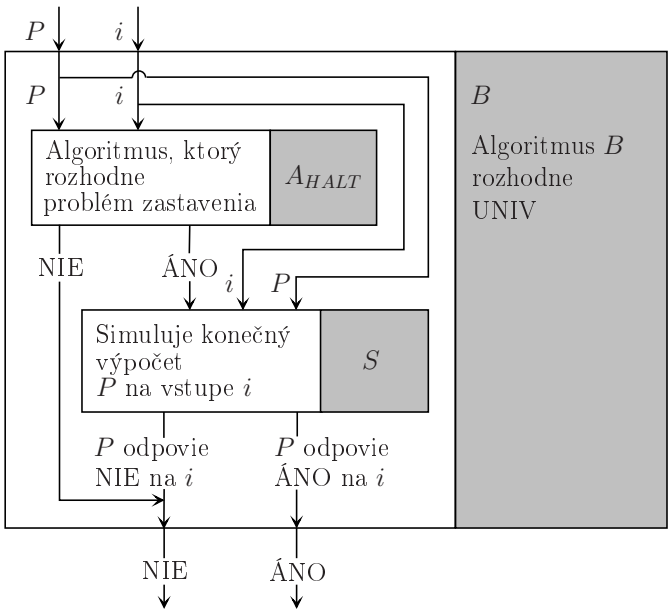
to znamená, že

> *vzhľadom na algoritmickú riešiteľnosť nie je UNIV ťažší ako HALT.*

Čo to znamená? Musíme ukázať, že existencia algoritmu pre HALT zaručuje algoritmus na riešenie UNIV. Predpokladajme teda, že máme algoritmus A_{HALT} pre HALT. Vytvoríme algoritmus B pre UNIV (obr. 4.18).

Algoritmus B pracuje pre vstup (P, i) nasledujúcim spôsobom:

1. B odovzdá svoj vstup (P, i) bez zmeny algoritmu A_{HALT}.
2. Algoritmus A_{HALT} rozhodne (v konečnom čase), či P pre i zastane alebo nie. A_{HALT} odpovie ÁNO, v prípade, že P pre i zastane. V inom prípade odpovie A_{HALT} NIE.



obr. 4.18

3. V prípade, že A_{HALT} odpovie NIE, vie B s určitosťou, že P neakceptuje číslo i (lebo P pre i pracuje nekonečne dlho) a odpovie NIE. („i nepatrí do $M(P)$“).

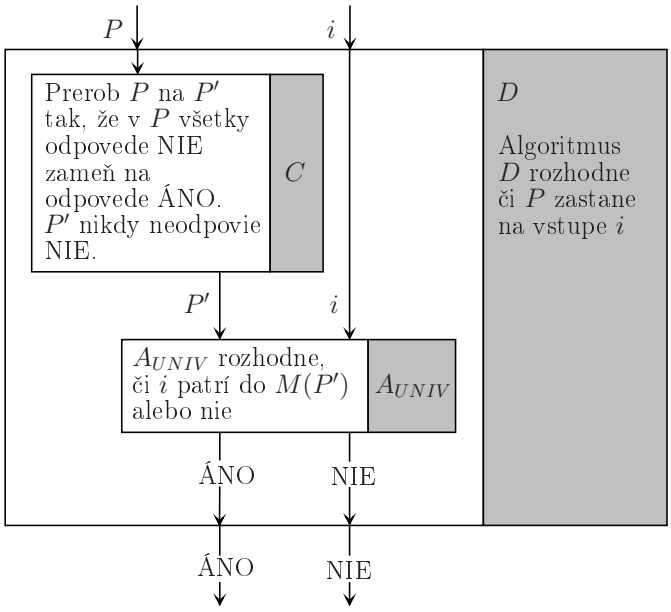
4. V prípade, že A_{HALT} odpovie ÁNO, simuluje B podprogramom S (obr. 4.18) konečnú prácu P pre i. Touto konečnou simuláciou B zistí, či P akceptuje číslo i, alebo nie a tento výsledok preberie za svoj vlastný (obr. 4.18).

Na základe konštrukcie priamo vidíme, že B rozhodne správne, či i patrí do $M(P)$ alebo nie. Ešte musíme overiť, či B pracuje vždy len konečne dlho. Podľa predpokladov je A_{HALT} algoritmus, a preto A_{HALT} odpovie v konečnom čase, to znamená, že v časti A_{HALT} sa nemôže B dostať do nekonečného výpočtu. Takže B vždy zastane, a teda B je algoritmus na riešenie univerzálneho problému.

Práve sme ukázali, že UNIV je ľahší alebo rovnako ťažší ako HALT. Chceme ukázať, že oba problémy sú rovnako ťažké. Z tohto dôvodu musíme ešte ukázať aj obrátený vzťah:

$$\text{HALT} \leq_{Alg} \text{UNIV}.$$

To znamená, že na základe algoritmickej riešiteľnosti UNIV chceme usúdiť



obr. 4.19

na algoritmickú riešiteľnosť HALT. Nech A_{UNIV} je algoritmus, ktorý rozhodne UNIV. Vytvoríme algoritmus D pre problém HALT, ktorý pre každý vstup (P, i) pracuje nasledujúco (obr. 4.19):

1. D odovzdá P podprogramu C, ktorý P prerobí na P' nasledujúcim spôsobom. C nájde v P všetky príkazy, ktorými sa dáva odpoveď „NIE“ a text „NIE“ v nich nahradí textom „ÁNO“. Takže P' nikdy nedá odpoveď NIE a platí:

   „Každý konečný výpočet P' skončí s odpoveďou ÁNO a P' akceptuje presne tie čísla i, pre ktoré P pracoval konečne dlho.“

2. D odovzdá P' a i algoritmu A_{UNIV} (obr. 4.19). A_{UNIV} rozhodne, či i patrí do $M(P')$ alebo nie.

3. D prevezme od A_{UNIV} odpoveď ÁNO alebo NIE a vyhlási ju za svoju vlastnú odpoveď.

Úloha 4.15 Vysvetlite čo najpodrobnejšie, prečo je D algoritmus na riešenie problému zastavenia.

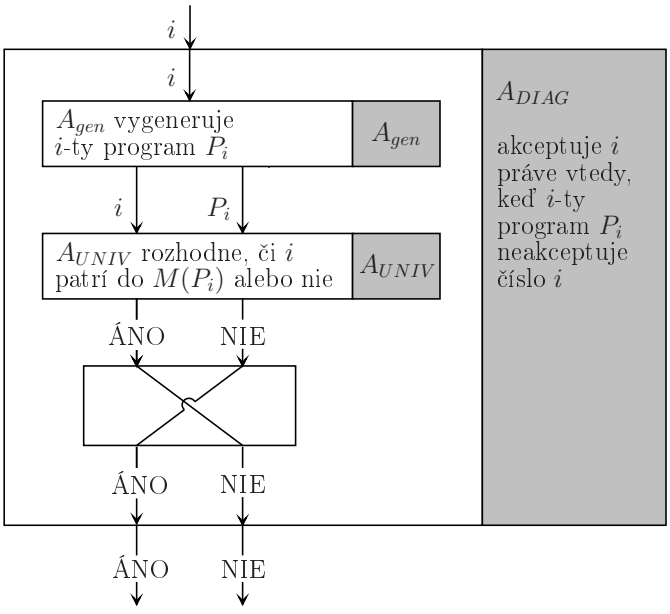
Úloha 4.16 (tvrdý oriešok) Redukcia UNIV $\leq_{Alg}$ HALT na obr. 4.18 a redukcia HALT $\leq_{Alg}$ UNIV (obr. 4.19) nevyzerajú rovnako. Často uprednostňu-

jeme druh redukcie znázornený na obr. 4.19, ktorý zodpovedá typickej redukcii v matematike. Vstup (prípad problému) (P, i) pre HALT zmeníme na (P', i) pre UNIV tak, že riešenie pre (P', i) problému A_{UNIV} môžeme priamo prebrať, ako riešenie prípadu problému (P, i) pre HALT. Schéma tejto redukcie je zobrazená na obr. 4.8 a obr. 4.19. Nájdite podobne jednoduchú redukciu na dokázanie toho, že UNIV $\leq_{Alg}$ HALT. To znamená, že musíte algoritmicky premeniť vstup (P, i) prípadu problému UNIV na vstup (P', i) problému HALT takým spôsobom, že budete môcť priamo prebrať odpoveď A_{HALT} pre (P', i) (riešenie (P', i) pre problém zastavenia) ako riešenie pípadu problému (P, i) pre UNIV.

Ukázali sme, že vzhľadom na algoritmickú riešiteľnosť sú univerzálny problém a problém zastavenia rovnako ťažké. To znamená, že buď sú oba problémy algoritmicky riešiteľné, alebo sú oba algoritmicky neriešiteľné. Ako sme už oznámili, máme v úmysle dokázať ich neriešiteľnosť. Na to stačí ukázať, že jeden z nich sa nedá vyriešiť ľahšie ako $(\mathbb{N}, M(\text{DIAG}))$. Ukážeme, že

$$(\mathbb{N}, M(\text{DIAG})) \leq_{Alg} \text{UNIV}.$$

Predpokladajme, že máme algoritmus A_{UNIV} na riešenie UNIV a s jeho pomocou vytvoríme algoritmus A_{DIAG} na rozhodnutie $(\mathbb{N}, M(\text{DIAG}))$. Algoritmus A_{DIAG} pre každé prirodzené číslo i odpovie ÁNO v prípade, že i-ty program P_i neakceptuje číslo i a odpovie NIE v prípade, že P_i akceptuje číslo i. A_{DIAG} pracuje pre každý vstup i nasledujúcim spôsobom:



obr. 4.20

1. A_{DIAG} pošle vstup i podprogramu A_{gen}, ktorý vytvorí i-ty program P_i a dá ho ako svoj výstup.

2. A_{DIAG} zoberie i a P_i a oba dá ako vstup A_{UNIV}. A_{UNIV} rozhodne, či P_i akceptuje číslo i (odpoveď „ÁNO“), alebo ho neakceptuje (odpoveď „NIE“).

3. A_{DIAG} odpovede A_{UNIV} navzájom vymení. V prípade, že A_{UNIV} odpovedal „ÁNO“ (i patrí do $M(P_i)$), potom i nepatrí do $M(\text{DIAG})$ a A_{DIAG} správne odpovie „NIE“. V prípade, že A_{UNIV} odpovedal „NIE“ (i nepatrí do $M(P_i)$), potom i patrí do $M(\text{DIAG})$ a A_{DIAG} musí odpovedať „ÁNO“.

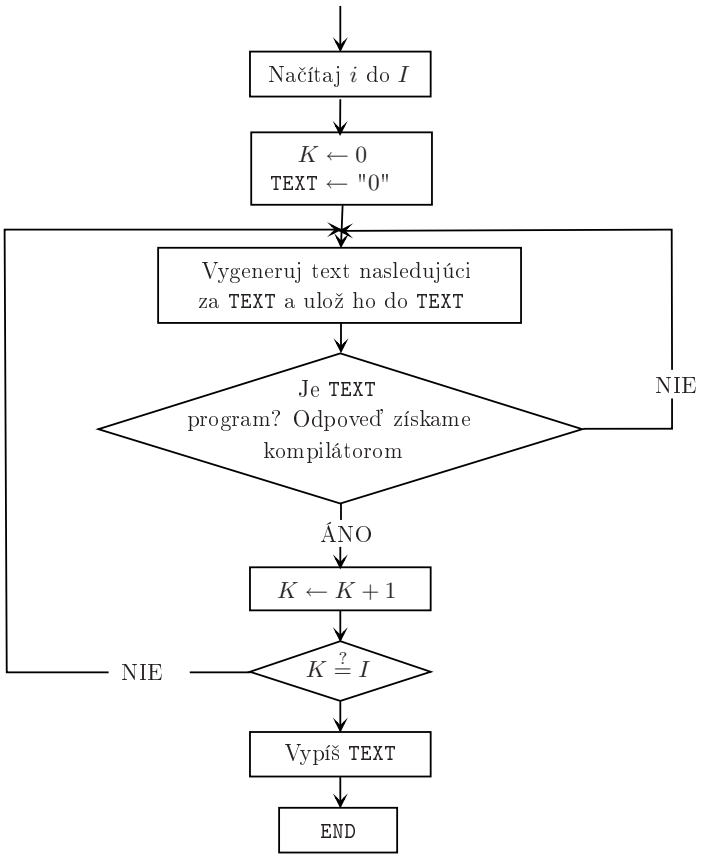
Z opisu práce A_{DIAG} pre i ihneď vidíme, že A_{DIAG} pracuje správne, keď pracujú správne A_{UNIV} a A_{gen}. To, že A_{UNIV} je algoritmus pre UNIV, sme predpokladali v rámci redukcie. Ostáva otázka, či skutočne vieme vytvoriť algoritmus A_{gen}, ktorý pre ľubovoľné číslo i vytvorí v konečnom čase text i-teho programu P_i. A_{gen} môže pracovať nasledujúcim spôsobom. Vytvára po sebe idúce texty vzhľadom na očíslovanie uvedené na začiatku kapitoly. Na každý text aplikuje kompilátor, ktorý preverí, či text predstavuje alebo nepredstavuje program. Pritom si A_{gen} eviduje počet kladných odpovedí. V okamihu, keď dosiahne i kladných odpovedí, vie, že posledný vytvorený program predstavuje i-ty program P_i. Schéma (vývojový diagram) programu A_{gen} je na obr. 4.21.

Úloha 4.17 Využitím redukcie $(\mathbb{N}, M(\text{DIAG}))$ na HALT ukážte, že platí $(\mathbb{N}, M(\text{DIAG})) \leq_{Alg} \text{HALT}$.

Úloha 4.18 Nech $M(\overline{\text{DIAG}})$ je množina všetkých prirodzených čísiel takých, že P_i akceptuje číslo i. Teda $M(\overline{\text{DIAG}})$ obsahuje presne tie prirodzené čísla, ktoré nie sú v $M(\text{DIAG})$. Ukážte prostredníctvom redukcie, že $(\mathbb{N}, M(\text{DIAG})) \leq_{Alg} (\mathbb{N}, M(\overline{\text{DIAG}}))$ a $(\mathbb{N}, M(\overline{\text{DIAG}})) \leq_{Alg} (\mathbb{N}, M(\text{DIAG}))$.

Ukázali sme, že rozhodovací problém $(\mathbb{N}, M(\text{DIAG}))$, univerzálny problém UNIV a problém zastavenia HALT nie sú algoritmicky riešiteľné. Pritom sú problémy UNIV a HALT dôležité pri testovaní programov, a teda majú praktický význam. Žiaľ, informatici vedia dokázať aj tvrdenia, ktoré nás sklamú, že všetky dôležité problémy testovania programov nie sú riešiteľné. Je to dokonca také zlé, že ani nasledujúca, na prvý pohľad ľahká úloha, nie je algoritmicky riešiteľná.

Nech je f_0 funkcia na prirodzených číslach, ktorej výsledok je 0 pre každý vstup i. Takéto funkcie nazývame konštantné funkcie, pretože výsledok je úplne nezávislý od vstupu (argumentov). Nasledujúci program



obr. 4.21

```
0 Výstup ← „0“
1 End,
```

ktorý sa vôbec na vstup i nepozrie (neprečíta ho), počíta funkciu f_0. Napriek tomu, neexistuje algoritmus, ktorý rozhodne, či zadaný program P počíta funkciu f_0[7]. Musíme tomu rozumieť tak, že v tomto rozhodovacom probléme môžeme dostať ako vstup aj veľmi dlhé programy, ktoré robia mnoho zbytočného alebo dokonca aj nezmyselného. Otázka ale je, či na koniec neodpovedia predsa len správny výsledok „0“.

Úloha 4.19 (tvrdý oriešok) Nech je $\mathcal{M}_0$ množina všetkých programov P, takých, že $M(P) = \emptyset$. Inak povedané $\mathcal{M}_0$ obsahuje všetky programy, ktoré na každý vstup odpovedia „NIE“ („0“), alebo počítajú nekonečne dlho. Dokážte, že nie

[7]Teda rovnakú, ako predtým uvedený program.

je algoritmicky rozhodnuteľné, či daný program patrí alebo nepatrí do $\mathcal{M}_0$ (teda, či daný program neakceptuje žiaden vstup).

V tejto kapitole sme sa naučili niečo dôležité. *Otázky ohľadom syntaxe a problémy typu „Je daný text programom?“ sú algoritmicky riešiteľné.* Pre dané prirodzené číslo i dokonca vieme skonštruovať program P_i. *Otázky ohľadom sémantiky, zisťujúce význam a správnosť programov sú algoritmicky neriešiteľné.*

4.5 Zhrnutie alebo čo z toho, čo sme objavili, bolo najdôležitejšie?

Zmarili sme nádej zo začiatku dvadsiateho storočia, že sa všetko dá automatizovať. Zistili sme, *že existujú problémy, ktoré sa nedajú riešiť pomocou strojov pracujúcich na základe algoritmov.* Toto tvrdenie platí nezávisle od súčasných alebo budúcich počítačových technológií.

K algoritmicky neriešiteľným problémom patria mnohé úlohy z praxe, ako napríklad:

- Je program korektný (počíta to, na čo bol vyvinutý)?
- Vykonáva program nekonečný výpočet (nekonečné opakovanie cyklu)?

V informatike vznikajú veľké výskumné tímy, ktoré nerobia nič iné, len skúmajú možnosti testovania programov[8]. Ukazuje sa, žiaľ, že aj *veľmi jednoduché úlohy testovania programov, ako napríklad „Počíta program konštantnú funkciu?“ sú algoritmicky neriešiteľné.* Výskumníci v tejto oblasti sú radi, keď sú schopní vytvoriť programy na testovanie aspoň čiastočnej správnosti programov. Pritom ide o testovanie obmedzených programov, ktoré sú špeciálnym spôsobom reprezentované, alebo o „vychytanie“ typických programátorských chýb bez akejkoľvek záruky, že objavíme všetky chyby.

Pri algoritmických úlohách alebo programoch rozlišujeme syntaktické a sémantické problémy. *Syntaktické* úlohy sa zaoberajú formálnou správnosťou zápisu programov v danom programovacom jazyku a väčšinou sú algoritmicky riešiteľné. *Sémantické* otázky sa zaoberajú významom programu. Napríklad:

[8] To potvrdzuje dôležitosť testovania programov v praxi.

- „Čo počíta daný program?“
- „Rieši vytvorený program daný problém?“
- „Skončí program pre zadaný vstup?“

Všetky netriviálne sémantické problémy týkajúce sa programov sú algoritmicky neriešiteľné.

Aby sme získali tieto vedomosti, naučili sme sa dve metódy používané na výskum a dokazovanie. Prvá z nich bola metóda diagonalizácie, ktorú sme využili už pri skúmaní nekonečnosti. Touto metódou sme ukázali, že je viac problémov ako programov, a preto musia existovať problémy, ktoré sú algoritmicky neriešiteľné. Náš prvý algoritmicky neriešiteľný problém bol rozhodovací problém $(\mathbb{N}, M(\mathrm{DIAG}))$, teda rozhodnutie príslušnosti k množine tvorenej prvkami na uhlopriečke (diagonále). Na rozšírenie algoritmickej neriešiteľnosti na ďalšie problémy sme použili metódu redukcie. Využívala sa už dlho na dosiahnutie pozitívneho cieľa, prenášanie riešiteľnosti úloh na ďalšie úlohy. Hlavnou myšlienkou je, že

$$\text{problém } P_1 \text{ nie je ťažší ako problém } P_2,\ P_1 \leq_{Alg} P_2,$$

keď pomocou algoritmu na riešenie P_2 vieme vytvoriť aj algoritmus na riešenie P_1. Vtedy hovoríme, že P_1 je *redukovateľný* na P_2.

V pozitívnom smere potom $P_1 \leq_{Alg} P_2$ implikuje výsledok, že z algoritmickej riešiteľnosti P_2 vyplýva algoritmická riešiteľnosť P_1. V negatívnom smere, ktorý sme použili, znamená $P_1 \leq_{Alg} P_2$, že z algoritmickej neriešiteľnosti P_1 vyplýva algoritmická neriešiteľnosť P_2. Metódu redukcie sme používali v negatívnom zmysle, z algoritmickej neriešiteľnosti $(\mathbb{N}, M(\mathrm{DIAG}))$ sme ukázali algoritmickú neriešiteľnosť problému zastavenia a univerzálneho problému.

Návody riešenia vybraných úloh

Úloha 4.3 Reálnych čísiel s konečnou reprezentáciou je presne $|\mathbb{N}|$. Vieme, že $|\mathbb{N}| \cdot 2 = |\mathbb{N}|$ a aj, že $|\mathbb{N}| \cdot |\mathbb{N}| = |\mathbb{N}|$. Pretože platí $|\mathbb{R}| > |\mathbb{N}|$, znamená to tiež, že

$$|\mathbb{R}| > |\mathbb{N}| \cdot |\mathbb{N}|.$$

Z tohto dôvodu, je jasné, že počet reálnych čísiel s konečnou reprezentáciou tvorí iba nekonečne menší zlomok množiny všetkých reálnych čísiel.

Úloha 4.6 Prvých 17 miest binárnej reprezentácie QUAD môžeme znázorniť najlepšie v nasledujúcej tabuľke.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
QUAD	1	1	0	0	1	0	0	0	0	1	0	0	0	0	0	0	1	0

Uvedenú tabuľku môžete rozšíriť na ľubovoľne veľa miest.

Úloha 4.7 Prvých 10 pozícií DIAG pre hypotetickú tabuľku na obr. 4.7 je

$$\text{DIAG} = 0101000011.$$

Úloha 4.8 Chceme ukázať, že

> $M(\text{2-DIAG})$ = množina všetkých párnych čísiel $2i$, takých, že $2i$ nepatrí do $M(P_i)$

je nerozpoznateľná. Hlavná myšlienka je veľmi podobná diagonalizácii na obr. 4.3. Vytvoríme 2-DIAG tak, že sa bude odlišovať od každého riadka tabuľky. Jediný rozdiel oproti DIAG je, že 2-DIAG sa líši od i-teho riadka na mieste $2i$ (namiesto i-teho, ako je to v DIAG). Najlepšie to môžeme znázorniť nasledujúcou tabuľkou na obr. 4.22.

	0	1	2	3	4	5	6	7	8	9	10	11	12	⋯
M(P_0)	**0**	0	1	1	0	1	1	0	1	1	1	1	0	
M(P_1)	1	0	**1**	1	0	0	0	0	1	0	1	1	0	
M(P_2)	1	1	1	1	**1**	1	1	1	0	0	0	1	0	
M(P_3)	0	1	0	1	0	0	**0**	0	1	1	1	0	0	
M(P_4)	1	0	1	0	1	0	1	0	**1**	0	1	0	1	
M(P_5)	0	1	0	1	1	0	0	1	0	1	**0**	1	1	
M(P_6)	0	0	0	0	0	0	0	0	0	0	0	0	**0**	⋯
⋮													⋮	⋱

obr. 4.22

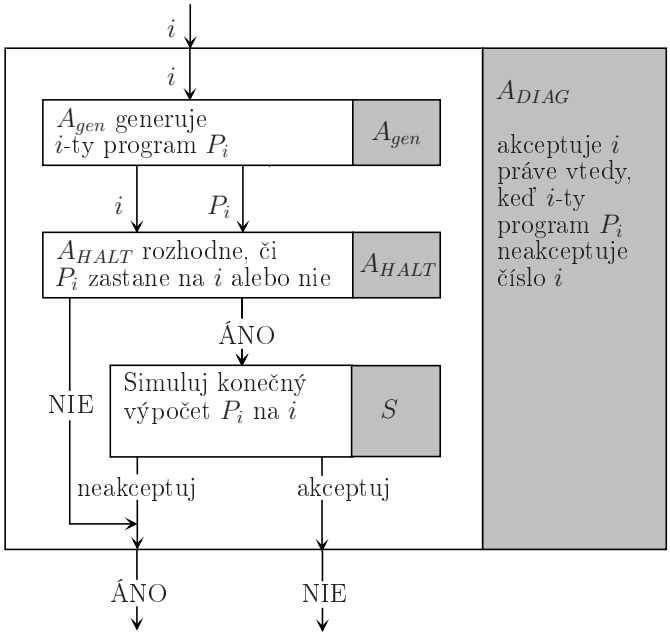
Pozície v rámčeku označujú priesečník i-teho riadka a $2i$-teho stĺpca, to znamená pozície, v ktorých sa 2-DIAG odlišuje od jednotlivých riadkov tabuľky. Teda prvých 13 pozícií 2-DIAG je uvedených v tabuľke na obr. 4.22:

$$\text{2-DIAG} = \underline{1}0\underline{0}0\underline{0}\underline{1}0\underline{0}\underline{1}\underline{0}1\ldots$$

Vidíme, že na každej nepárnej pozícii sú v 2-DIAG nuly, ktoré nezohrávajú pri rozpoznateľnosti 2-DIAG nijakú úlohu. Podčiarknuté párne pozície (začíname nultou pozíciou) zodpovedajú orámčekovaným pozíciám na obr. 4.22. Takže 1 na začiatku zaručuje, že 2-DIAG neleží v nultom riadku, 0 na druhej pozícii zaručuje, že 2-DIAG neleží v prvom riadku, atď. Jednotka na 12. pozícii v 2-DIAG zaručuje, že 2-DIAG neleží v šiestom riadku tabuľky.

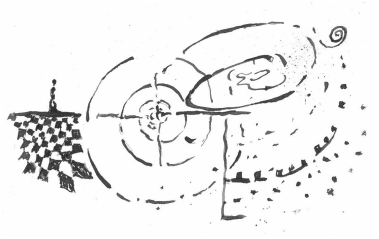
Úloha 4.17 Máme ukázať, že s hypotetickým algoritmom A_{HALT} pre HALT sa dá rozpoznať diagonálna množina $M(\text{DIAG})$. Začneme podobne ako na obr. 4.20

pri redukcii $(\mathbb{N}, M(\text{DIAG})) \leq_{Alg}$ UNIV. Pre zadané číslo i musíme rozhodnúť, či $i \in M(\text{DIAG})$, to znamená, či P_i neakceptuje číslo i. Takže najprv prostredníctvom A_{gen} vytvoríme program P_i a algoritmom A_{HALT} sa spýtame, či sa P_i pre vstup i zastaví alebo nezastaví (obr. 4.23).



obr. 4.23

Keď sa P_i pre vstup i nezastaví, potom i nepatrí do $M(P_i)$ a s istotou vieme, že $i \notin M(P_i)$ (P_i neakceptuje i), správna odpoveď je teda ÁNO ($i \in M(\text{DIAG})$). Keď sa P_i pre i zastaví, potom postupujeme podobne ako na obr. 4.18. Simulujeme výpočet P_i pre vstup i a upravíme výsledok simulácie. Keď P_i neakceptuje i, akceptujeme i. Keď P_i akceptuje i, neakceptujeme číslo i (obr. 4.23).



Niet väčšej škody ako stratený čas

Michelangelo Buonarotti

Kapitola 5

Teória zložitosti alebo čo môžeme robiť, keď na výpočet nestačí energia celého vesmíru?

5.1 Nie je riešiteľné ako riešiteľné alebo úvod do teórie zložitosti

V kapitole 4 sme zistili, že sú zaujímavé úlohy, ktoré algoritmicky nevieme riešiť. Dokonca sme sa naučili, ako sa dá ukázať, že niektoré problémy sú v algoritmickom zmysle neriešiteľné. Na začiatku šesťdesiatych rokov dominovala v základnom výskume klasifikácia (rozdelenie) algoritmických problémov na algoritmicky riešiteľné a algoritmicky neriešiteľné. Situácia sa postupne menila s čoraz širším nasadením počítačov v civilných oblastiach. Stále častejšie sa počítače používali na plánovanie, optimalizáciu pracovných procesov a simulovanie drahých výskumných experimentov. Prví programátori a návrhári algoritmov museli čeliť tvrdej realite. Napísali programy, zadali ich do počítača a všetci v miestnosti sa potili, pretože počítač sa v pravom zmysle slova zahrieval a chladenie bolo v tých časoch problém. Len výsledkov sa nie a nie dočkať. Na počítačoch sa musela často robiť údržba[1], a tak sa dalo počítať len medzi dvoma údržbami. Tento čas nestačil na úspešné dokončenie výpočtov.

[1] nezriedka aj denne

Informatici nedokázali vyriešiť predkladané úlohy napriek tomu, že tieto boli očividne algoritmicky riešiteľné. Vyvinuli predsa algoritmy na riešenie zadaných úloh, ktoré premenili na programy. Boli potrebné predpovede, koľko času budú algoritmy vyžadovať na prácu. Tak prišiel na rad opäť najdôležitejší proces - tvorba pojmov. Vznikli pojmy výpočtová zložitosť a v určitom zmysle aj zložitosť algoritmických problémov. Čoskoro sme rozpoznali, čo znamená počítať efektívne. Mnohé algoritmy sa nedali použiť a to nie preto, že na vypočítanie výsledku nestačilo zopár dní. Na uskutočnenie výpočtu by nám nestačili ani miliardy rokov. Takže takéto algoritmy neboli prakticky použiteľné. Niekto by mohol povedať: „V poriadku, poďme hľadať teda pre dané problémy efektívnejšie algoritmy.“ Lenže máme stovky problémov (úloh), pre ktoré sa aj napriek veľkej vynaloženej námahe nepodarilo nájsť efektívne algoritmy. Opäť sa vynárajú principiálne otázky:

> *Je to len naša neschopnosť a nedostatok vedomostí o algoritmike, pre ktorú nevieme nájsť efektívny spôsob riešenia niektorých problémov?*
>
> *Alebo existujú algoritmicky riešiteľné problémy, na ktorých riešenie nie je efektívny algoritmus (existujú problémy, ktoré nie sú prakticky riešiteľné napriek tomu, že sú algoritmicky riešiteľné)?*

Tieto otázky viedli k rozvoju teórie zložitosti, ktorá sa v prvom rade pokúša merať stupeň obťažnosti algoritmických úloh vzhľadom na ich výpočtovú zložitosť. Jej hlavným cieľom je rozdelenie algoritmicky riešiteľných úloh na prakticky (efektívne) riešiteľné a na prakticky neriešiteľné. Teória zložitosti ukazuje, že sú problémy, pri ktorých by nestačila na výpočet ich riešenia (na uskutočnenie potrebných výpočtov) ani celá energia vesmíru.

> *Algoritmicky (automaticky) riešiteľné neznamená teda ešte prakticky riešiteľné.*

Rozpoznať, čo je prakticky riešiteľné a vývoj efektívnych algoritmov, je dodnes najťažšie a najdôležitejšie jadro výskumu v teoretickej informatike.

Dôkazy a argumentácia sú v tejto oblasti často také obťažné, že všetko, s čím sme sa doteraz oboznámili, je oproti tomu ako detská hra. Preto sa nepokúsime v tejto kapitole uviesť podrobne technické detaily. Našťastie ich nepotrebujeme na pochopenie divov, ktoré predstavíme neskoršie. Jediné, čo k tomu budeme potrebovať, je pochopiť jednotlivé koncepty a vý-

dobytky teórie zložitosti a schopnosť správne si vysvetliť ich význam. Tomuto cieľu sa venuje táto kapitola.

5.2 Ako meriame zložitosť výpočtov?

Pojem zložitosti v zmysle množstva výpočtovej práce je pre výpočtové vedy centrálny a v rebríčku dôležitosti informatických pojmov určite nasleduje hneď po už skôr zavedených pojmoch, akými sú program a algoritmus. Keď postupujeme matematicky (presne), mali by sme sa najprv dohodnúť na matematickom modeli algoritmov, a potom môžeme merať vynaloženú výpočtovú prácu, ako počet vykonaných operácií v tomto modeli. Našťastie je presný spôsob merania zložitosti zväčša potrebný len pri odvodení kvantitatívnych zákonov spracovania informácií, ktorým sa tu nebudeme zaoberať kvôli vysokému stupňu obťažnosti. Pri bežnej práci (návrhu a implementácii) algoritmov nám často stačí nasledujúci jednoduchý spôsob merania zložitosti, ktorým vo väčšine prípadov dostaneme vierohodné výsledky.

Ako sa dá jednoducho merať zložitosť algoritmu? Nech je A algoritmus na riešenie nejakého problému (úlohy) U. Najprv musíme povedať, čo je to zložitosť A pre prípad I problému U[2]. Najjednoduchší spôsob merania je definovať **zložitosť algoritmu A pre prípad I problému U** ako

> *počet vykonaných operácií počítača pri výpočte A na I.*

Pretože predpokladáme, že sa operácie vykonávajú jedna po druhej, hovoríme presnejšie o **časovej zložitosti algoritmu A pre prípad I**. Keďže časová zložitosť je pre nás najdôležitejšou mierou na meranie a posúdenie efektívnosti algoritmov, často pre ňu používame len skrátené označenie zložitosť. Druhou najdôležitejšou mierou zložitosti je pre nás **pamäťová zložitosť algoritmu A pre prípad I**, ktorá označuje

> *počet použitých premenných, teda počet pamäťových miest (registrov) pri výpočte A na I.*

Všimnime si napríklad úlohu vypočítať hodnotu kvadratického polynómu

$$a \cdot x^2 + b \cdot x + c$$

pre zadané čísla

$$a = 3,\ b = 4,\ c = 5 \text{ a } x = 7.$$

[2] Informatici používajú termín inštancia problému.

Naivný algoritmus môže počítať nasledovne:

$L \leftarrow b \cdot x$

{vynásob b s x a výsledok ulož do premennej (na pamäťové miesto, ktorého meno je) L}

$X \leftarrow x \cdot x$

$Y \leftarrow a \cdot X$

{teraz je v Y uložená hodnota ax^2}

$D \leftarrow L + c$

{teraz je v D uložená hodnota $b \cdot x + c$}

$R \leftarrow Y + D$

{teraz je v R uložený výsledok $ax^2 + bx + c$}

Hneď vidíme, že pre zadané čísla a, b, c a x sa zrealizuje nasledujúcich päť aritmetických operácií.

$b \cdot x \to L$	$x \cdot x \to X$	$a \cdot X \to Y$	$L + c \to D$	$Y + D \to R$
$4 \cdot 7 = 28$	$7 \cdot 7 = 49$	$3 \cdot 49 = 147$	$28 + 5 = 33$	$147 + 33 = 180$

Takže časová zložitosť A pre $I = (a = 3, b = 4, c = 5, x = 7)$ je presne 5. Keď sa sústredíme na pamätanie si hodnôt pre a, b, c, x, I, X, Y, D a R, vidíme, že pamäťová zložitosť je presne 9. Všimnite si, že konkrétne hodnoty premenných a, b, c a x nemali vplyv na zložitosť algoritmu. Preto hovoríme, že časová zložitosť A je presne 5 pre každý prípad problému (pre každý kvadratický polynóm).

Úloha 5.1 Zapíšte algoritmus A v programovacom jazyku z 2. kapitoly, ktorý poskytuje iba jednoduché strojové inštrukcie. Myslite aj na načítanie hodnôt pre a, b, c a x.

a) Aká je časová zložitosť A v tejto konkrétnej implementácii?
b) Ste schopní prepísať program tak, aby ste v ňom použili menej ako 9 registrov?

Postup algoritmu A môžeme znázorniť aj nasledovnou reprezentáciou polynómu:

$$a \cdot x \cdot x + b \cdot x + c \ .$$

Hneď vidíme tri násobenia a dve sčítania, ktoré musíme vykonať. Pokúsime sa algoritmus vylepšiť. Podľa známeho distributívneho zákona platí

$$a \cdot x \cdot x + b \cdot x = (a \cdot x + b) \cdot x,$$

s jeho pomocou dostaneme nasledujúcu reprezentáciu kvadratického polynómu:

$$ax^2 + bx + c = (ax + b) \cdot x + c \ .$$

V novej reprezentácii musíme vykonať len dve násobenia a dve sčítania, takže zložitosť výsledného vylepšeného algoritmu je 4.

Úloha 5.2 Uvažujeme polynóm štvrtého stupňa:

$$f(x) = a_4 \cdot x^4 + a_3 \cdot x^3 + a_2 \cdot x^2 + a_1 \cdot x + a_0 \ .$$

Vysvetlite, ako sa dá vypočítať hodnota polynómu len s použitím štyroch násobení a štyroch sčítaní.

Úloha 5.3 (tvrdý oriešok) Navrhnite algoritmus, ktorý spočíta pre zadané hodnoty $a_0, \ldots, a_n$ a x, hodnotu každého polynómu n-tého stupňa

$$a_n \cdot x^n + a_{n-1} \cdot x^{n-1} + \ldots + a_2 \cdot x^2 + a_1 \cdot x + a_0$$

s použitím najviac n násobení a n sčítaní.

Vidíme, že množstvo práce, ktorú musíme vykonať, závisí od našej šikovnosti pri návrhu algoritmu. Ešte presvedčivejším príkladom je výpočet hodnoty x^{16} pre dané x. Keď rozpíšeme x^{16} ako

$$x \cdot x \cdot x \cdot x \cdot x \cdot x \cdot x \cdot x \cdot x \cdot x \cdot x \cdot x \cdot x \cdot x \cdot x \cdot x,$$

vidíme, že na výpočet hodnoty x^{16} týmto spôsobom potrebujeme 15 operácií. Nasledujúce vyjadrenie:

$$x^{16} = (((x^2)^2)^2)^2$$

nám dáva efektívnejšiu metódu

$$\begin{array}{llll} x^2 = x \cdot x & x^4 = x^2 \cdot x^2 & x^8 = x^4 \cdot x^4 & x^{16} = x^8 \cdot x^8 \\ L \leftarrow x \cdot x & L \leftarrow L \cdot L & L \leftarrow L \cdot L & L \leftarrow L \cdot L \end{array}$$

výpočtu x^{16}, ktorej stačia len 4 násobenia.

Úloha 5.4 Vypočítajte:

a) x^6 s 3 násobeniami,

b) x^{64} so 6 násobeniami,
c) x^{18} s 5 násobeniami,
d) x^{45} s 8 násobeniami.

Dá sa spočítať x^{45} s menej ako 8 násobeniami?

Ale situácia, ktorá bola pri výpočte hodnoty kvadratického polynómu, že zložitosť je rovnaká pre každý prípad polynómu, je dosť netypická. Nastala preto, lebo problém je jednoduchý a v určitom zmysle sme merali veľmi hrubo. Naše meranie zložitosti je v poriadku, v prípade keď sú všetky vstupné hodnoty a, b, c a x čísla, z ktorých sa každé dá bez problémov zapamätať v jednom 16 alebo 32 bitovom registri. Čo sa ale stane, ak majú čísla veľkosť niekoľko sto bitov? V takom prípade nemôžeme považovať množstvo práce na vykonanie aritmetickej operácie s obrovským číslom za také isté, aké sú nevyhnutné na vykonanie operácie so 16-bitovým číslom, ktorá je k dispozícii v hardvéri počítača. V niektorých aplikáciách sa naozaj používajú také veľké čísla, vtedy je nevyhnutné napísať program, ktorý počíta operácie s veľkými číslami a využíva pri tom len operácie s číslami bežnej veľkosti, ktoré sú k dispozícii. Čitateľa nebudeme zaťažovať týmto technickým problémom a budeme predpokladať, že naše čísla neprekročia rozumnú veľkosť. Na základe takéhoto predpokladu budeme merať časovú zložitosť ako počet aritmetických operácií, operácií porovnávania a podobných základných operácií počítača alebo programovacieho jazyka.

Aj pri takomto predpoklade nie je typické, že algoritmus pre všetky vstupy (prípady problému) vyžaduje stále rovnaké množstvo práce. Keď máme utriediť podľa abecedy telefónny zoznam obce s 3 000 obyvateľmi a mesta s 500 000 obyvateľmi, bude množstvo práce očividne veľmi rôzne. To nie je nič prekvapujúce, ale dostávame sa tým k jadru veci. Očakávame, že zložitosť bude závisieť od **veľkosti vstupu**. Čo je to veľkosť vstupu, lepšie povedané ako ju meriame, je na nás. Pri triedení to môže byť napríklad počet (3 000 alebo 500 000 mien obyvateľov) objektov, ktoré máme usporiadať. Pri výpočte hodnoty ľubovoľného polynómu, môžeme brať ako veľkosť vstupu stupeň polynómu (teda maximálny možný počet koeficientov mínus 1). V tomto prípade je polynóm

$$a_n x^n + a_{n-1} x^{n-1} + \ldots + a_2 x^2 + a_1 x + a_0$$

reprezentovaný ako vstup $n + 2$ číslami

$$(a_n, a_{n-1}, a_{n-2}, \ldots, a_2, a_1, a_0, x)$$

a povieme, že tento prípad problému má veľkosť n. Naivný algoritmus A počíta hodnotu polynómu stupňa n nasledujúcim spôsobom:

$$\underbrace{x \cdot x = x^2}_{\text{1.násobenie}} \qquad \underbrace{x \cdot x^2 = x^3}_{\text{2. násobenie}} \qquad \cdots \qquad \underbrace{x \cdot x^{n-1} = x^n}_{(n-1)\text{-vé násobenie}}$$

$$\underbrace{a_1 \cdot x}_{n\text{-té násobenie}} \qquad \underbrace{a_2 \cdot x^2}_{(n+1)\text{-vé násobenie}} \qquad \cdots \qquad \underbrace{a_n \cdot x^n}_{(2n-1)\text{-vé násobenie}}$$

a potom

$$a_0 \quad + \quad a_1 x \quad + \quad \cdots \quad + \quad a_n x^n$$
$$\uparrow \qquad\qquad \uparrow \qquad\qquad\qquad \uparrow$$
$$\text{1. sčítanie} \qquad \text{2. sčítanie} \qquad n\text{-té sčítanie}$$

Takže časová zložitosť A je funkcia

$$\check{\text{C}}\text{as}_A(n) = \underbrace{2n-1}_{\text{násobení}} + \underbrace{n}_{\text{sčítaní}} = 3n - 1 \ .$$

Teda **časová zložitosť Čas$_A$ algoritmu A** je

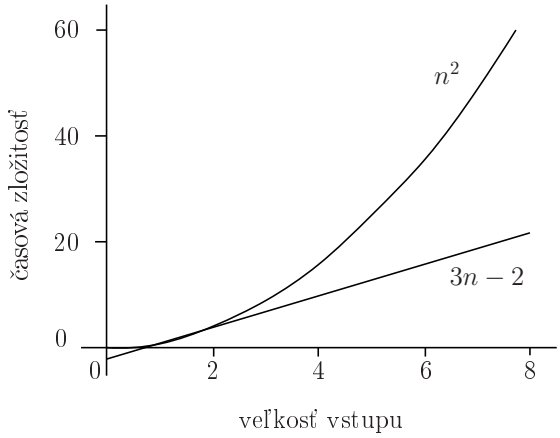
> *funkcia veľkosti vstupu, ktorá určuje počet Čas$_A(n)$ operácií algoritmu A nevyhnutných a postačujúcich na vyriešenie každého prípadu problému veľkosti n.*

Môže sa stať, že rôzne vstupy rovnakej veľkosti vyžadujú rôzne náklady. V takom prípade zoberieme Čas$_A(n)$ ako časovú zložitosť na najťažšom vstupe veľkosti n, teda ako maximum zložitostí A pre všetky vstupy veľkosti n. A sme v suchu, a preto sa pre túto definíciu rozhodli aj výskumníci. Takto máme hodnotou Čas$_A(n)$ zaručené, že algoritmus A úspešne zvládne vyriešiť pomocou Čas$_A(n)$ operácií každý prípad problému veľkosti n, a že existuje aspoň jeden vstup veľkosti n, pre ktorý A vykoná presne Čas$_A(n)$ operácií.

5.3 Načo slúži meranie zložitosti algoritmov?

Rovnako ako aj v iných vedeckých disciplínach slúžia nadobudnuté vedomosti aj na predpovedanie vývoja v rôznych situáciách a činnostiach, ktoré nás zaujímajú. Keď prostredníctvom takzvanej analýzy zložitosti určíme časovú zložitosť algoritmu, vieme pre zadané prípady problému vopred odhadnúť čas práce algoritmu bez toho, aby sme ho nechali na nich vykonávať. Okrem toho takýmto spôsobom môžeme porovnávať, aké

sú dobré (efektívne) dva alebo aj viac algoritmov pre tú istú úlohu. Väčšinou to robíme tak, že si nakreslíme ich funkcie zložitosti (časovej alebo pamäťovej).



obr. 5.1

Na obr. 5.1 sú v súradnicovej sústave nakreslené dve funkcie $3n-2$ a n^2. Na x-ovej osi je veľkosť vstupu a na y-ovej osi analyzovaná časová zložitosť. Ihneď vidíme, že pre všetky vstupy väčšie ako 3, je algoritmus s časovou zložitosťou $3n-2$ efektívnejší ako algoritmus so zložitosťou n^2. Výpočtom ľahko dokážeme, že

$$3n - 2 < n^2$$

je pre všetky prirodzené čísla väčšie ako 2.

Pozrime sa na iný príklad. Nech sú A a B dva algoritmy pre problém U, pre ktoré

$$\text{Čas}_A(n) = 2n^2 \text{ a } \text{Čas}_B(n) = 40n + 7\,000 \ .$$

Pretože lineárne funkcie ako $\text{Čas}_B(n)$ rastú pomalšie ako kvadratické funkcie akou je $\text{Čas}_A(n)$, natíska sa otázka, od akej veľkosti vstupu je pre nás výhodnejší algoritmus B ako algoritmus A. Odpovieme výpočtom. Otázka je teda, pre aké kladné celé číslo platí

$$2n^2 > 40n + 7\,000.$$

Táto otázka je ekvivalentná s otázkou (od oboch strán odpočítame celé číslo $40n + 7\,000$), keď platí

$$2n^2 - 40n - 7\,000 > 0.$$

Keď nerovnicu vydelíme 2 (obe strany zmenšíme na polovicu), dostaneme

$$n^2 - 20n - 3\,500 > 0 \ . \tag{5.1}$$

Teraz môžeme známou metódou riešenia kvadratických rovníc nájsť riešenia rovnice $n^2 - 20n - 3\,500 = 0$, alebo si jednoducho všimneme, že

$$(n + 50) \cdot (n - 70) = n^2 - 20n - 3\,500 \ .$$

V oboch prípadoch dostaneme riešenia (takzvané korene) -50 a 70 a vidíme, že (5.1) platí pre $n < -50$ a $n > 70$. Pretože nás zaujímajú len kladné celé čísla, skončíme s výsledkom:

(i) B je výhodnejší ako A pre prípady problémov, ktoré sú väčšie ako 70.

(ii) A a B sú rovnako dobré pre vstupy veľkosti $n = 70$.

(iii) A je priaznivejší ako B pre vstupy veľkosti 1 až 69.

Úloha 5.5 Uvažujte nad tromi algoritmami A, B a C, ktoré riešia rovnakú úlohu. Nech sú $\text{Čas}_A(n) = n^3/2 + 1$, $\text{Čas}_B(n) = n^2 + 7$ a $\text{Čas}_C(n) = 5n + 140$. Pokúste sa zistiť výpočtom a načrtnutím grafu funkcií, ktorý z uvedených troch algoritmov je najlepší, pre ktoré veľkosti vstupov.

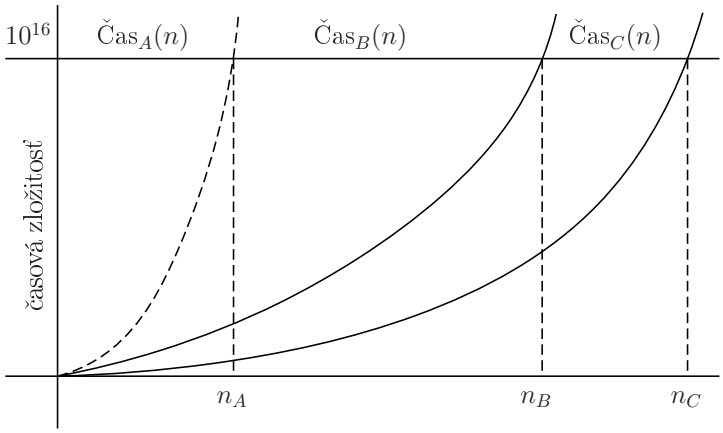
Iný druh otázok je nasledujúci. Používateľ vie celkom presne, že na výsledok môže čakať najviac určitý čas. Pre interaktívnu aplikáciu môže byť hranica už krátkych $10s$. Keď máme dobrý počítač, ktorý je schopný vykonať za sekundu 10^9 operácií, môžeme vykonať výpočty vyžadujúce najviac $10 \cdot 10^9 = 10^{10}$ operácií. Používateľovi ponúkneme algoritmus so zložitosťou $5n^3$. On si spočíta:

$$\begin{array}{rcll} 5n^3 & < & 10^{10} & | : 5 \\ n^3 & < & 2 \cdot 10^9 & | \sqrt[3]{} \text{ oboch strán} \\ n & \leq & 1\,250 & \end{array}$$

Používateľ týmto vie, že algoritmus môže úspešne použiť na vstupy veľkosti do 1250. Používateľ zvyčajne vie vstupy akých veľkostí sa typicky vyskytujú a môže sa hneď rozhodnúť, či nechá algoritmus implementovať, alebo bude požadovať rýchlejší algoritmus.

Teraz predpokladajme, že máme optimalizačnú úlohu, pri ktorej máme vybrať najlepšie z veľkého množstva riešení. Pritom ide o veľké investície, ako napríklad výstavba cestnej alebo železničnej siete, alebo o umiestnenie vysielačov na nejakom území. Pretože sa chceme dobre rozhodnúť, sme

ochotní venovať výpočtu veľa času a použiť drahú výpočtovú techniku. Takýmto spôsobom prídeme veľmi rýchlo na hranice uskutočniteľného, povedzme 10^{16} operácií. Vizuálne si hranicu môžeme znázorniť vodorovnou priamkou $y = 10^{16}$ ako na obr. 5.2. Tam, kde dosiahne časová zložitosť $\check{\text{C}}\text{as}_A(n)$ algoritmu A našu hranicu, môžeme na x-ovej osi prečítať veľkosť vstupu n_A. Takto vieme, pre ktoré veľkosti vstupov je použiteľný algoritmus A a pretože je známa veľkosť zadaného problému, ihneď vieme aj rozhodnúť, či je algoritmus pre nás vhodný.



obr. 5.2

Aby sme si ukázali dôležitosť efektívnosti algoritmu, analyzujeme situáciu v prípade niekoľkých časových funkcií. Nech $\check{\text{C}}\text{as}_A(n) = 3n - 2$. Potom vypočítame:

$$\begin{array}{rcll} 3n-2 & \leq & 10^{16} & | + 2 \text{ k obom stranám} \\ 3n & \leq & 10^{16}+2 & | \text{ vydelíme 3} \\ n & \leq & \frac{1}{3}(10^{16}+2) = n_A. & \end{array}$$

Vidíme, že o takom veľkom vstupe neuvažujeme, teda algoritmus A môžeme vždy použiť. Pre algoritmus B so zložitosťou $\check{\text{C}}\text{as}_B(n) = n^4$ dostaneme:

$$\begin{array}{rcll} n^4 & \leq & 10^{16} & |\sqrt[4]{} \text{ oboch strán} \\ n & \leq & (10^{16})^{1/4} = 10^4 & = 10\,000. \end{array}$$

Takže B je použiteľný pre vstupy do veľkosti $n_B = 10\,000$. Pretože typicky veľkosť vstupu pre väčšinu (ale nie všetky) vstupy nedosiahne n_B, B sa dá považovať za dobrý algoritmus.

Zoberme, že platí $\text{Čas}_C(n) = 10^n$. Potom to vyzerá nasledovne

$$\begin{aligned} 10^n &\leq 10^{16} \quad | \log_{10} \text{ oboch strán} \\ n &\leq 16, \end{aligned}$$

čo je zlé. Napriek obrovskému počtu operácií, ktorý máme k dispozícii, vieme úlohu vyriešiť len pre malé prípady. Keď si všimneme exponenciálnu funkciu $f(n) = 2^n$, vidíme, že

$$2^{n+1} = 2 \cdot 2^n,$$

takže zväčšenie veľkosti vstupu o 1 má za následok zdvojnásobenie výpočtových nákladov. Z toho môžeme usúdiť, že algoritmy s exponenciálnou časovou zložitosťou majú len veľmi obmedzené použitie.

Úloha 5.6 Predpokladajme, že informatici vylepšili algoritmus C so zložitosťou $\text{Čas}_C(n) = 10^n$ na algoritmus D so zložitosťou $\text{Čas}_D(n) = 4 \cdot (1.2)^n$. Ako sa zväčší rozsah problémov, ktoré môžeme spracovať, keď analyzujeme časovú hranicu 10^{16}?

Úloha 5.7 Predpokladajme, že máme k dispozícii počítač, ktorý vie vykonať 10^9 operácií za sekundu. Počet sekúnd, ktoré uplynuli od veľkého tresku je menší ako 10^{18}. Sme pripravení čakať 10^{18} sekúnd. Aké veľké problémy vieme spracovať algoritmom A, keď

(i) $\text{Čas}_A(n) = 10 \cdot n^2$?

(ii) $\text{Čas}_A(n) = 50 \cdot n^3$?

(iii) $\text{Čas}_A(n) = 2^n$?

(iv)* $\text{Čas}_A(n) = n! = n \cdot n(n-1) \cdot (n-2) \cdot \ldots \cdot 2 \cdot 1$? (tvrdý oriešok)

5.4 Hranice praktickej riešiteľnosti

V predchádzajúcej časti sme videli, ako vplýva časová zložitosť algoritmov na ich použiteľnosť. Náš cieľ je ale trocha náročnejší. Chceme merať obťažnosť algoritmických problémov, aby sme vedeli rozhodnúť, či sú alebo nie sú prakticky riešiteľné. Na jednej strane sa zdá, že cesta od merania zložitosti algoritmov k meraniu zložitosti problémov bude krátka a jasná. Ponúka sa nasledujúca definícia:

> *Zložitosť problému U je zložitosť najlepšieho (optimálneho vzhľadom na časovú zložitosť) algoritmu pre U.*

Hoci definícia vyzerá rozumne, nie je všeobecne požiteľná. Výskumníci ukázali, že sú úlohy, pre ktoré sa nedá určiť najlepší algoritmus. Pre takúto úlohu U sa dá každý algoritmus pre U podstatne[3] vylepšiť.

Pretože vo všeobecnosti sa nedá zložitosť každého problému U identifikovať s (najlepším) algoritmom pre U, hovoríme v informatike o hornom a dolnom odhade zložitosti problému.

Definícia 5.1 Nech je U problém a nech A je algoritmus, ktorý rieši U. Potom hovoríme, že časová zložitosť $\check{\text{C}}\text{as}_A(n)$ algoritmu A je **horným odhadom časovej zložitosti pre U**. O funkcii f hovoríme, že $f(n)$ je **dolným odhadom časovej zložitosti pre U**, keď neexistuje algoritmus B pre U, pre ktorý

$$\check{\text{C}}\text{as}_B(n) \leq f(n)$$

pre skoro všetky[4] n.

Táto definícia obťažnosti problémov stačí na objavenie niektorých dôležitých skutočností. Napríklad, že existujú ľubovoľne ťažké algoritmické problémy. Presnejšie povedané, pre ľubovoľne rýchlo rastúcu funkciu ako napríklad $2^n, n!$, ba aj 2^{2^n} sa dajú nájsť problémy, ktoré sú riešiteľné s takouto zložitosťou, ale s menšou nie. Takéto extrémne ťažké (prakticky neriešiteľné) problémy sa našli náročným použitím diagonalizačnej metódy, teda väčšinou sú to umelo vytvorené problémy.

Pre praktické problémy je určenie zložitosti oveľa zložitejšie. Poznáme tisíce úloh s exponenciálnymi hornými odhadmi, napríklad 2^n (lebo najlepšie známe algoritmy na ich riešenie potrebujú obrovské množstvo počítačovej práce). Na druhej strane nie sme schopní ukázať, že požiadavky na výpočtovú zložitosť ich riešenia sú vyššie ako lineárne (teda nemáme dolný odhad tvaru napr. $n \cdot \log n$ alebo n^2). Inými slovami, máme hŕbu problémov s obrovskou medzerou medzi ich dolným $c \cdot n$ a horným odhadom ako 2^n a nie sme schopní ich zložitosť určiť presnejšie. Informatici a matematici majú za sebou vyše štyridsať rokov neúspešných pokusov, pričom problém je očividne v obťažnosti určiť dolný odhad zložitosti konkrétnych úloh.

Dnes považujeme dôkaz dolného odhadu, a tým aj neexistencie efektívneho algoritmu, za najtvrdšie jadro celej informatiky. Je dokázané, že niektoré dolné odhady, ktoré by sme si želali vedieť sa dnešnými matematickými metódami ani nedajú dokázať. Teda je potrebný podstatný pokrok

[3] pre nekonečne veľa vstupov
[4] pre všetky až na konečne veľa

v rozvoji metód dokazovania v matematike, aby sme vedeli presnejšie určiť vyššie dolné odhady zložitosti konkrétnych algoritmických úloh.

Výskum zložitosti algoritmov a problémov nastolilo novú a v súčasnosti hlavnú otázku algoritmiky:

Ktoré algoritmické problémy sú prakticky riešiteľné?
Kde sú hranice algoritmickej riešiteľnosti?

Keď sa pozrieme na naše skúmanie v časti 5.3 a tabuľku 5.1, vidíme, že exponenciálne algoritmy v nijakom prípade nemôžeme označiť ako praktické. V tabuľke 5.1 uvádzame počet operácií pre 5 funkcí určujúcich zložitosť $10n$, $2n^2$, n^3, 2^n a $n!$ a pre vstupy 4 veľkostí 10, 50, 100 a 200.

n / $f(n)$	10	50	100	300
$10n$	100	500	1000	3000
$2n^2$	200	5000	20000	180000
n^3	1000	125000	1000000	27000000
2^n	1024	16 cifier	31 cifier	91 cifier
$n!$	$\approx 3.6 \cdot 10^6$	65 cifier	158 cifier	615 cifier

Tabuľka 5.1

Keď je počet operácií priveľký, píšeme namiesto čísla len počet jeho desiatkových cifier. Okamžite vidíme, že exponenciálne rýchlo rastúce funkcie zložitosti ako 2^n a $n!$ sú prakticky nepoužiteľné už pre vstupy malého rozsahu nad 50.

Po mnohých rokoch uvažovania sa informatici dohodli na nasledujúcej charakterizácii praktickej riešiteľnosti:

Algoritmus A, ktorého $Čas_A(n) \leq c \cdot n^d$ pre nejaké konštanty (konkrétne čísla) c a d nazývame ***polynomiálny algoritmus****.*
Každý problém, ktorý vieme vyriešiť polynomiálnym algoritmom, považujeme za ***prakticky riešiteľný****. Triedu všetkých rozhodovacích problémov[5], ktoré sú prakticky riešiteľné, označujeme* ***P****.*

Nebolo vôbec jednoduché všeobecne akceptovať túto definíciu. Dnes ju nepovažujeme, a tým ani polynomiálnu časovú zložitosť, ako ostrú hranicu medzi prakticky riešiteľným a prakticky neriešiteľným, ale len ako

[5]Na pripomenutie: rozhodovacie problémy majú výsledok ÁNO alebo NIE. Rozhodne sa teda, či vstup má, alebo nemá požadovanú vlastnosť (pozri kapitolu 4.3).

priblíženie sa k nášmu prvému pokusu ju určiť. K akceptovaniu tejto hranice nás vedú dva celkom odlišné dôvody – praktický a teoretický.

1. **Praktický dôvod**

   Je založený na skúsenostiach z vývoja algoritmov. Nepraktickosť exponenciálnych algoritmov bola každému jasná. Analýzy a prax nás naučili, že algoritmy s časovou zložitosťou do n^3 a za určitých podmienok až do n^6 sú použiteľné. Ale algoritmus s časovou zložitosťou n^{100} je pre vstupy reálnych veľkostí ešte menej prakticky použiteľný ako algoritmus s časovou zložitosťou 2^n, pretože $n^{100} > 2^n$, pre takmer všetky rozumné veľkosti vstupu n. Môžeme teda nazvať problém, pre ktorý najlepší algoritmus beží v čase n^{100} prakticky riešiteľný? Skúsenosti s reálnymi problémami ukazujú, že takéto problémy sa v praxi nevyskytujú. Keď sa našiel polynomiálny algoritmus, pričom bol stupeň polynómu vysoký, potom sa takmer vždy podarilo nájsť na riešenie rovnakého problému iný algoritmus, ktorý pracoval v čase menšom ako $c \cdot n^6$, alebo ešte častejšie v menšom ako $c \cdot n^3$, pre nejakú konštantu c. Je len málo výnimiek problémov riešiteľných v polynomiálnom čase, ktoré nie sú prakticky riešiteľné. Preto nie je z praktického pohľadu trieda P príliš veľká a problémy z P sú považované za prakticky riešiteľné.

2. **Teoretický dôvod**

   Definícia dôležitej triedy, akou sú prakticky riešiteľné problémy, musí byť robustná v tom zmysle, že bude nezávislá od definície použitého výpočtového modelu. Nesmie sa stať, že problém je prakticky riešiteľný z pohľadu programovacieho jazyka JAVA, ale nie z pohľadu iného modelu, alebo iného programovacieho jazyka. To by bol prípad, keby sme sa pokúsili definovať triedu prakticky riešiteľných problémov, ako takých, ktorých horný odhad časovej zložitosti je $c \cdot n^6$. V teórii používané výpočtové modely často majú na simuláciu programu v bežnom programovacom jazyku n^2 krát vyššie nároky na čas. Takže problém sa môže dať v jazyku JAVA vyriešiť v čase n^3 a v nejakom inom modeli by potreboval čas n^5. Pojem polynomiálneho algoritmu, a tým aj triedy P, je ale dostatočne robustný. Trieda polynomiálne riešiteľných problémov je rovnaká pre všetky rozumné modely. Teda dôkaz príslušnosti, alebo neprísluš nosti do triedy P je od výpočtového modelu nezávislý, a teda všeobecne platný a môže slúžiť z teoretického hľadiska na klasifikáciu problémov na riešiteľné a prakticky neriešiteľné.

5.5 Ako rozoznáme ťažký problém?

Hlavnou úlohou teórie zložitosti je klasifikácia konkrétnych algoritmických problémov vzhľadom na ich výpočtovú zložitosť. Návrhom algoritmov dostávame horné odhady zložitosti problémov, ale týmto spôsobom nie sme schopní odvodiť dolný odhad zložitosti konkrétnych problémov. Ako ich teda môžeme klasifikovať? V skutočnosti to v absolútnom zmysle nemôžeme. Robíme to, čo obyčajne robia vedci a ľudia zo zdravým sedliackym rozumom v podobných situáciách. Nepôjdeme hlavou proti múru a namiesto určovania presnej zložitosti znovu a znovu pre každý prípad problému, uspokojíme sa s hodnoverným, hoci aj nie stopercentným odhadom zložitosti.

Čo znamená „hodnoverne“ odôvodniť, že neexistuje polynomiálny algoritmus, ktorý rieši daný problém? Dnes je známych vyše 4000 zaujímavých úloh, pre ktoré sa napriek veľkej námahe nenašiel polynomiálny algoritmus. Neúspešná námaha v prípade jednotlivých problémov ale nie je ešte dostatočný dôvod na to, aby sme ich vyhlásili za prakticky neriešiteľné. Bolo by to aj nesprávne. Pre problémy lineárneho programovania a testovania prvočíselnosti sme sa neúspešne pokúšali mnoho rokov[6] nájsť polynomiálne algoritmy. A bolo obrovskou udalosťou, keď sa pre ne našli polynomiálne algoritmy. A hoci na základe skúseností sme skôr verili, že existujú, ukazuje to jednoznačne, aké by mohlo byť nebezpečné, vyhlásiť ich za ťažké (prakticky neriešiteľné) na základe roky trvajúcich neúspešných pokusov ich efektívne riešiť.

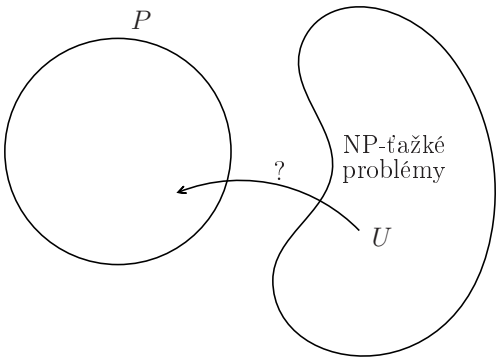
Na začiatku sedemdesiatych rokov, na základe opakovaných negatívnych skúseností s mnohými problémami, formulovali nezávisle na sebe S.A. Cook a L.A. Levin nasledovnú hodnovernú definíciu

> *Problém U je ťažký (nie je v P, alebo je prakticky neriešiteľný), keď existencia polynomiálneho algoritmu pre U by súčasne znamenala existenciu polynomiálnych algoritmov pre tisíce problémov, doteraz považovaných za ťažké (pre ktoré nie sme schopní nájsť efektívny algoritmus).*

Celé si to môžeme dobre predstaviť pomocou obr. 5.3. Vľavo je trieda P polynomiálne riešiteľných problémov. Vpravo je trieda s tisíckami problémov, pre ktoré sme nenašli polynomiálny algoritmus. Teraz si predstavte, že sme našli efektívny algoritmus pre jeden problém U z tejto triedy. V dôsledku našej definície ťažkých problémov by to znamenalo efektívne

[6] v prípade testov prvočíselnosti dokonca tisíce rokov

vyriešenie všetkých problémov z tejto triedy.



obr. 5.3

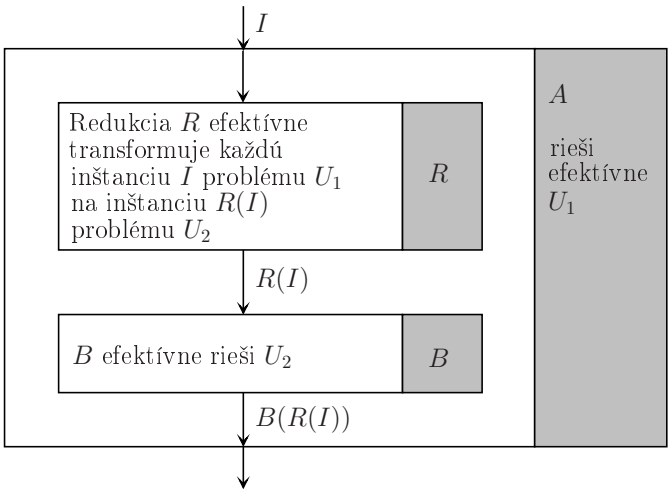
Teraz je o mnoho uveriteľnejšie, že problém U je naozaj ťažký. Na základe doterajších skúseností takmer nik dnes neverí, že problémy, ktoré považujeme podľa našej definície za ťažké, by sa dali efektívne vyriešiť. Nevieme si predstaviť, že by sme boli takí hlúpi a napriek mnohým pokusom by sme nevedeli nájsť efektívne riešenie nijakého z mnoho tisíc problémov, hoci pre všetky takéto riešenia existujú.

Odborníci nazývajú tieto problémy, ktorých efektívne riešenie by automaticky znamenalo efektívne riešenie ďalších tisícov problémov, ktoré považujeme za ťažké, **NP-ťažké** problémy. Otázkou ostáva:

„Ako ukážeme, že nejaký konkrétny problém je NP-ťažký?“

Opäť nám pomôže metóda redukcie. Používali sme ju, aby sme ukázali, že algoritmická riešiteľnosť jedného problému znamená algoritmickú riešiteľnosť iného problému. V tomto prípade stačí nahradiť vlastnosť *„algoritmicky“* vlastnosťou *„efektívne algoritmicky“*. To vieme dosiahnuť prostredníctvom modelu efektívnej redukcie. Ako vidíme na obr. 5.4, problém U_1 efektívne redukujeme na problém U_2 tak, že nájdeme efektívny algoritmus R, ktorý každý prípad problému U_1 prevedie na ekvivalentný prípad $R(I)$ problému U_2.

Pod ekvivalentným tu rozumieme, že riešenie prípadu $R(I)$ problému U_2, je rovnaké ako prípadu I problému U_1. Teda výsledok $B(R(I))$ výpočtu B pre $R(I)$ môžeme automaticky prevziať ako výsledok pre I. Príklad takejto redukcie sme uviedli už na obr. 4.8 v 4. kapitole. Redukovali sme tam efektívne problém riešenia kvadratickej rovnice na problém riešenia normovanej kvadratickej rovnice. Dôsledok bol, že efektívne riešenie



obr. 5.4

normovanej kvadratickej rovnice znamenalo efektívne riešenie všeobecnej kvadratickej rovnice.

Hovoríme, že algoritmus R na obr. 5.4 je **polynomiálna redukcia U_1 na U_2** , keď je R polynomiálny algoritmus s vlastnosťou:

> *Pre všetky prípady I problému U_1 je riešenie I problému U_1 zároveň riešením prípadu $R(I)$ pre U_2.*

Keď sú U_1 aj U_2 rozhodovacie problémy, znamená to, že buď je pre oba prípady I a $R(I)$ správna odpoveď ÁNO, alebo je pre obe správna odpoveď NIE. V tomto prípade hovoríme tiež, že prípad $R(I)$ problému U_2 je **ekvivalentný** s prípadom I problému U_1.

Keď máme polynomiálnu redukciu R problému U_1 na U_2, hovoríme tiež, že

> **U_1 je redukovateľný v polynomiálnom čase na U_2**

a píšeme

$$U_1 \leq_{pol} U_2.$$

Podobne ako pri všeobecnej redukcii, znamená $U_1 \leq_{pol} U_2$, že U_2 nie je ľahšie ako U_1 vzhľadom na riešiteľnosť v polynomiálnom čase. Teda buď sú oba U_1 aj U_2 rovnako ťažké (oba sa dajú efektívne riešiť, alebo sa ani jeden nedá efektívne riešiť), alebo je U_1 efektívne riešiteľný a U_2 nie je. Vylúčená je jedine situácia, že by bol U_2 efektívne riešiteľný a U_1 by nebol.

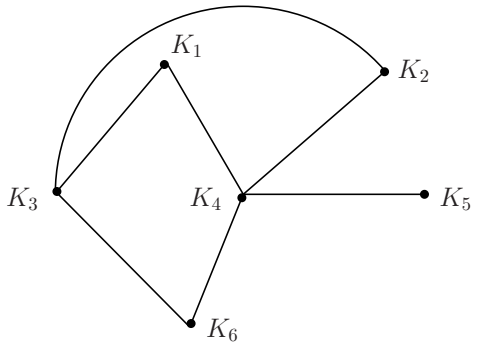
Úloha 5.8 Ukážte, že úloha vypočítať výšku rovnostranného trojuholníka so známou dĺžkou strany sa dá polynomiálne redukovať na úlohu vypočítať dĺžku strany v pravouhlom trojuholníku (Pytagorova veta).

Úloha 5.9 Nech je U_2 úloha vyriešiť lineárnu rovnicu tvaru $a + bx = 0$. Nech je U_1 úloha vyriešiť lineárnu rovnicu tvaru $a + bx = c + dx$. Ukážte, že $U_1 \leq_{pol} U_2$.

Pojem polynomiálnej redukovateľnosti sa veľmi rýchlo stal úspešným nástrojom na klasifikáciu problémov. Dnes poznáme tisíce problémov, ktoré sa dajú v oboch smeroch vzájomne jeden na druhý redukovať. A $U_1 \leq_{pol} U_2$ a súčasne $U_2 \leq_{pol} U_1$ znamená, že U_1 a U_2 sú oba rovnako ťažké v zmysle, že buď sú oba riešiteľné v polynomiálnom čase, alebo sa ani jeden nedá efektívne riešiť. Na základe metódy redukcie poznáme tisíce algoritmických problémov, ktoré sú buď všetky efektívne riešiteľné, alebo sa nedá efektívne riešiť ani jeden z nich a tieto problémy nazývame NP-ťažké[7].

Príklady redukcií, ktoré sme doteraz uviedli, neilustrovali dostatočne použitie redukčnej metódy, pretože šlo očividne o ľahké úlohy. Teraz si preto ukážeme redukciu medzi dvoma NP-ťažkými problémami.

Ako problém U_1 si zoberieme **problém stráženia**, ktorý sa v odbornej reči nazýva problém pokrytia vrcholov a označuje sa **VC**. Máme danú sieť ulíc s n križovatkami (odborne sa nazývajú „vrcholy“) a s ulíc spájajúcich križovatky[8] (obr. 5.5).



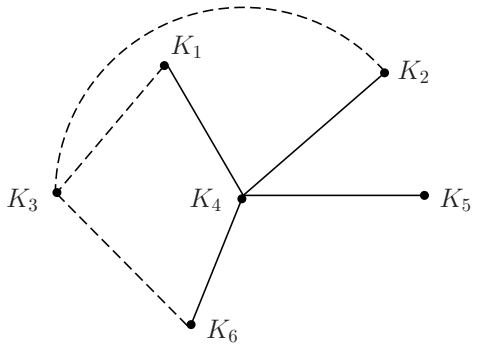
obr. 5.5

Križovatky sme na obr. 5.5 nakreslili ako čierne body, označili sme ich K_1, K_2, K_3, K_4, K_5 a K_6. Čiary medzi bodmi sú ulice. Na obr. 5.5 je 7 ulíc.

[7] Neuvádzame formálnu definíciu NP-ťažkého problému, pretože využíva koncepty, ktoré sme nezaviedli.

[8] Koniec slepej ulice považujeme tiež za križovatku (K_5 na obr. 5.5).

Ulice môžeme tiež označiť. Ulicu medzi K_1 a K_2 označíme $Ulica(K_1, K_2)$. Pretože ulice nemajú predpísaný smer, označuje $Ulica(K_2, K_1)$ rovnakú ulicu ako $Ulica(K_1, K_2)$. Na križovatku môžeme umiestniť pozorovateľa. Predpokladáme, že pozorovateľ je schopný strážiť všetky ulice vychádzajúce z tejto križovatky, pričom každú ulicu vidí až po susednú križovatku. Takže pozorovateľ z K_3 dokáže strážiť tri ulice $Ulica(K_3, K_1)$, $Ulica(K_3,K_2)$ a $Ulica(K_3, K_6)$. Na obr. 5.6 sú prerušovanou čiarou nakreslené ulice, ktoré je vidno z K_3.



obr. 5.6

Súčasťou úlohy je ešte zadané číslo m. Otázka je, či na dohľad nad všetkými ulicami postačuje m pozorovateľov. Presnejšie, dá sa rozmiestniť m pozorovateľov na križovatky tak, aby boli všetky ulice pod dohľadom? Pre cestnú sieť na obr. 5.5 stačia dvaja pozorovatelia. Keď je pozorovateľ na K_4, prezerá štyri ulice $Ulica(K_4, K_1)$, $Ulica(K_4, K_2)$, $Ulica(K_4, K_5)$ a $Ulica(K_4, K_6)$. Pozorovateľ na K_3 prezerá zvyšné tri ulice.

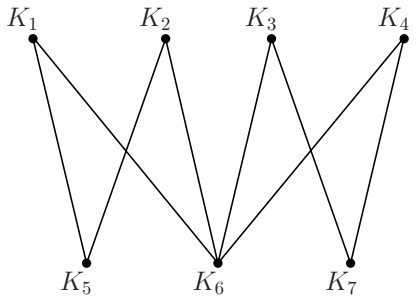
Úloha 5.10 Predpokladajme, že v cestnej sieti na obr. 5.5 nesmieme dať pozorovateľa na K_4. Koľko pozorovateľov potrebujeme v tomto prípade?

Úloha 5.11 Pozorujme cestnú sieť na obr. 5.7. Stačia traja pozorovatelia na to, aby sme mali pod dohľadom všetky ulice?

Úloha 5.12 Do obr. 5.7 dokreslite novú ulicu $Ulica(K_1, K_2)$ medzi K_1 a K_2. Koľko pozorovateľov potrebujeme v tomto prípade, aby sme mali pod dohľadom celú sieť?

Úloha 5.13 K cestnej sieti na obr. 5.5 pripojte dve ulice $Ulica(K_5, K_6)$ a $Ulica(K_2, K_5)$. Stačia 3 pozorovatelia na dohľad nad celou sieťou?

Ako druhý problém U_2 zoberme úlohu **LIN(0,1)** čo je určenie, či existuje riešenie systému lineárnych nerovníc nad booleovskými premennými.



obr. 5.7

Napríklad

$$x_1 + 2x_2 - 3x_3 + 7x_4 \geq 3$$

je lineárna nerovnica so 4 neznámymi x_1, x_2, x_3 a x_4. Hovoríme, že ide o nerovnosť nad booleovskými premennými, keď neznáme x_1, x_2, x_3 a x_4 môžu mať hodnoty len 0 a 1. Prípad problému LIN(0,1) je napríklad

$$\begin{aligned} x_1 + 2x_2 + x_3 + x_4 &\geq 3 \\ x_1 + x_4 &\geq 0 \\ 2x_1 + x_2 - x_3 &\geq 1. \end{aligned}$$

To je systém 3 lineárnych nerovníc so 4 neznámymi. Úlohou je rozhodnúť pre systém lineárnych nerovníc, či existuje taká voľba hodnôt 0 a 1 pre neznáme, že všetky nerovnice budú súčasne splnené. V našom prípade stačí napríklad zvoliť x_1 a x_2 rovné 1 ($x_1 = x_2 = 1$) a x_3 a x_4 rovné 0 ($x_3 = x_4 = 0$). Vidíme, že

$$\begin{aligned} x_1 + 2x_2 + x_3 + x_4 = 1 + 2 \cdot 1 + 0 + 0 = 3 &\geq 3 \\ x_1 + x_4 = 1 + 0 = 1 &\geq 0 \\ 2x_1 + x_2 - x_3 = 2 \cdot 1 + 1 - 0 = 3 &\geq 1, \end{aligned}$$

teda všetky nerovnice sú splnené.

Úloha 5.14 Nájdite iné booleovské hodnoty premenných x_1, x_2, x_3 a x_4 nášho systému tak, aby boli splnené všetky tri nerovnice.

Úloha 5.15 Má riešenie nasledujúci systém lineárnych nerovníc?

$$\begin{aligned} x_1 + x_2 - 3x_3 &\geq 2 \\ x_1 - 2x_2 - x_4 &\geq 0 \\ x_1 + x_3 + x_4 &\geq 2 \end{aligned}$$

Teraz ukážeme, že platí

$$\text{VC} \leq_{pol} \text{LIN}(0,1),$$

teda, že sa dá problém pozorovateľov VC polynomiálne redukovať na problém lineárnych nerovníc s booleovskými premennými.

Na to musíme efektívne skonštruovať pre každý prípad problému VC „ekvivalentný“ prípad problému LIN(0,1). Vysvetlíme postup redukcie na príklade cestnej siete N na obr. 5.5. Nech je $(N, 3)$ prípad problému VC, ktorý zodpovedá otázke, či na pozorovanie siete N na obr. 5.5 stačia 3 pozorovatelia. Aby sme túto otázku premenili na otázku o lineárnych nerovniciach, zvolíme si 6 booleovských premenných x_1, x_2, x_3, x_4, x_5 a x_6. Premennú x_i priradíme ku križovatke K_i a bude mať nasledujúci význam:

$x_i = 1$ znamená, že na K_i je pozorovateľ,
$x_i = 0$ znamená, že na K_i sa nenachádza pozorovateľ.

Napríklad hodnoty $x_1 = 1, x_2 = 0, x_3 = 1, x_4 = 0, x_5 = 1, x_6 = 0$ znamenajú, že máme troch pozorovateľov, ktorí sa nachádzajú na križovatkách K_1, K_3 a K_5 a na križovatkách K_2, K_4 a K_6 nemáme pozorovateľov.

Ako prvú vyjadríme prostredníctvom lineárnej nerovnice

$$x_1 + x_2 + x_3 + x_4 + x_5 + x_6 \leq 3$$

skutočnosť, že môžeme mať najviac troch pozorovateľov. V prípade, že sieť N je pod kontrolou pozorovateľov, je pod dohľadom každá ulica. Aby sme to zabezpečili, musí byť na každej ulici $Ulica(K_i, K_j)$, spájajúcej križovatky K_i a K_j, pozorovateľ aspoň na jednej z križovatiek K_i a K_j. Pre ulicu $Ulica(K_1, K_4)$ musí byť aspoň jeden pozorovateľ na križovatke K_1 alebo K_4. To vyjadríme lineárnou nerovnicou:

$$x_1 + x_4 \geq 1.$$

Táto nerovnosť je splnená, keď $x_1 = 1$ alebo $x_4 = 1$ a to je presne riešenie, ktoré potrebujeme. Keď zoberieme do úvahy všetkých 7 ulíc (K_1, K_3), (K_1, K_4), (K_2, K_3), (K_2, K_4), (K_3, K_6), (K_4, K_5), (K_4, K_6) siete N, dostaneme nasledujúci systém 8 lineárnych nerovníc $L1$:

$$\begin{aligned} x_1 + x_2 + x_3 + x_4 + x_5 + x_6 &\leq 3 \ \{\text{najviac 3 pozorovatelia}\} \\ x_1 + x_3 &\geq 1 \ \{Ulica(K_1, K_3) \text{ pod dohľadom}\} \\ x_1 + x_4 &\geq 1 \ \{Ulica(K_1, K_4) \text{ pod dohľadom}\} \\ x_2 + x_3 &\geq 1 \ \{Ulica(K_2, K_3) \text{ pod dohľadom}\} \end{aligned}$$

$$\begin{aligned} x_2 + x_4 &\geq 1 \ \{Ulica(K_2, K_4) \text{ pod dohľadom}\} \\ x_3 + x_6 &\geq 1 \ \{Ulica(K_3, K_6) \text{ pod dohľadom}\} \\ x_4 + x_5 &\geq 1 \ \{Ulica(K_4, K_5) \text{ pod dohľadom}\} \\ x_4 + x_6 &\geq 1 \ \{Ulica(K_4, K_6) \text{ pod dohľadom}\} \end{aligned}$$

Teraz platí: Systém $L1$ má riešenie presne vtedy, keď existuje riešenie prípadu $(N, 3)$ problému stráženia (teda keď traja pozorovatelia stačia na dohľad nad celou sieťou N). Platí dokonca viac: Každé riešenie $L1$ dáva riešenie $(N, 3)$. Napríklad riešením pre $L1$ je

$$x_1 = 0, x_2 = 1, x_3 = 1, x_4 = 1, x_5 = 0, x_6 = 0,$$

lebo máme len troch pozorovateľov (prvá nerovnosť), $x_2 = 1$ garantuje splnenie štvrtej a piatej nerovnice, $x_3 = 1$ garantuje splnenie druhej, štvrtej a šiestej nerovnosti a $x_4 = 1$ garantuje splnenie tretej, piatej, siedmej a ôsmej nerovnosti. Tým je splnených všetkých 8 nerovností. Zároveň hneď vidíme, že traja pozorovatelia umiestnení na križovatkách K_2, K_3 a K_4 majú pod dohľadom celú sieť N.

Úloha 5.16

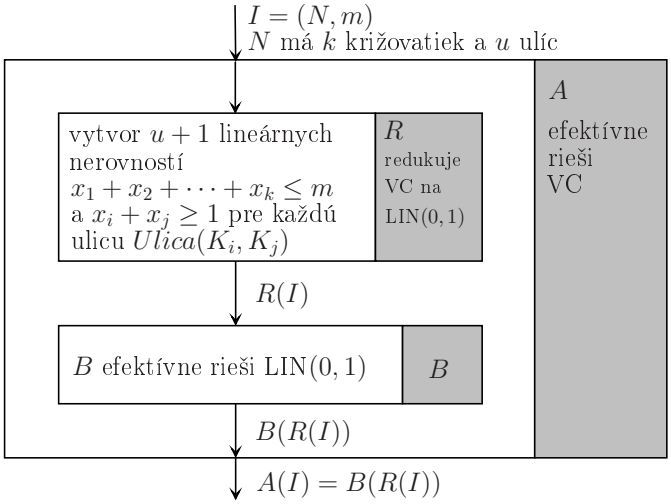
a) Nájdite všetky riešenia systému $L1$ (priradenia hodnôt 0 a 1 premenným $x_1, x_2, \ldots, x_6$) a im zodpovedajúce riešenia pre $(N, 3)$.
b) Existuje riešenie, pri ktorom $x_4 = 0$ (na križovatke K_4 nie je pozorovateľ)? Vysvetlite, prečo je to tak.
c) Rozšírte sieť N o ulicu $Ulica(K_3, K_4)$. Novú sieť označme N'. Má $(N', 2)$ riešenie?

Všeobecný opis redukcie VC na LIN(0,1) je na obr. 5.8.

Vidíme, že redukcia R sa dá veľmi jednoducho realizovať programom. Efektívnosť R garantuje, že z existencie efektívneho algoritmu B pre problém LIN(0,1) vyplýva existencia efektívneho algoritmu pre VC.

Úloha 5.17 Uvažujte nad prípadom $(N, 3)$ problému VC pre sieť na obr. 5.7. Využite vyššie opísanú redukciu R na to, aby ste vytvorili ekvivalentný systém lineárnych rovníc. Nájdite všetky riešenia systému lineárnych rovníc, a tým aj zodpovedajúce riešenia $(N, 3)$.

Úloha 5.18 Zostrojte sieť N' tak, že sieť N na obr. 5.7 rozšírite o dve ulice $Ulica(K_2, K_3)$ a $Ulica(K_5, K_6)$. Ako vyzerá systém lineárnych rovníc pre $(N', 4)$? Má riešenie?



obr. 5.8

Ukázali sme, že VC $\leq_{pol}$ LIN$(0, 1)$. Dá sa ukázať aj LIN$(0, 1)$ $\leq_{pol}$ VC. Táto redukcia je ale príliš technická, preto ju nebudeme vysvetlovať.

Takže VC a LIN(0,1) sú rovnako ťažké v tom zmysle, že existencia polynomiálneho algoritmu pre jeden z problémov znamená existenciu polynomiálneho algoritmu pre druhý z nich. Dnes je známych vyše 4000 takýchto rovnako ťažkých problémov a pre nijaký z nich nepoznáme efektívny algoritmus. Z tohto dôvodu veríme, že ide ozaj o ťažké problémy a redukovateľnosť v polynomiálnom čase používame na klasifikáciu problémov na ľahké (riešiteľné v polynomiálnom čase) a ťažké (neriešiteľné v polynomiálnom čase). Aby sme nový problém U považovali za ťažký, stačí ukázať, že

$$U' \leq_{pol} U$$

pre niektorý problém U' z ťažkej triedy. Lebo keď platí $U' \leq_{pol} U$, efektívny algoritmus pre U by znamenal automaticky, že máme efektívne algoritmy aj pre tisíce problémov, ktoré doteraz boli považované za ťažké.

Informatici zvolili túto cestu ako základ pre určenie obťažnosti konkrétnych algoritmických problémov, pretože neboli schopní dokazovať dolné odhady ich zložitosti. Nedá sa celkom vylúčiť, že raz niekto príde s geniálnym nápadom a efektívne vyrieši všetky NP-ťažké problémy. Odvolávajúc sa na veľké skúsenosti a presvedčenie väčšiny bádateľov, smieme teda veriť, že sú tieto problémy naozaj ťažké? Je na to pragmatická odpoveď. Aj keby existovali efektívne algoritmy na riešenie NP-ťažkých

problémov, tieto problémy budú pre nás ťažké až dovtedy, kým niekto tieto algoritmy neobjaví. Takže je jedno, ako to v skutočnosti naozaj je, v súčasnosti nemáme pre tieto problémy efektívne metódy, a preto ich považujeme za ťažké. Teoretici v oblasti výpočtovej zložitosti ale majú ďalšie závažné argumenty, ktoré podporujú ich vieru o obťažnosti NP-ťažkých problémov. Dokázali, že efektívne metódy riešenia NP-ťažkých problémov by znamenali, že objavenie dôkazov matematických tvrdení je rovnako ťažké ako preverenie, či sú dané dôkazy korektné. A dnes nik neverí, že vytváranie dôkazov (jedna z najťažších intelektuálnych činností matematika) nie je ťažšie, ako kontrola správnosti už existujúceho dôkazu.

Teraz by sa mohol čitateľ vecne spýtať: Keď sú informatici takí presvedčení o tom, že pre NP-ťažké problémy neexistujú polynomiálne algoritmy, prečo to jednoducho nevyhlásia za novú axiómu informatiky? Odpoveď je jednoznačná: nesmú to urobiť. Za axiómy môžeme postulovať len tvrdenia a definície, ktoré nie sú dokázateľné. Ako sme vysvetlili v 1. kapitole, neexistuje možnosť dokazovať axiómy, tie možno len vyvrátiť. Keď vyvinieme techniky na dokazovanie dolných odhadov zložitosti, môže sa stať, že budeme vedieť dokázať skutočnú obťažnosť NP-ťažkých problémov. Preto slepo neveríme v obťažnosť NP-ťažkých problémov a venujeme mnoho úsilia tomu, aby sme ju zdôvodnili. Tento problém patrí nielen k nemnohým ústredným problémom informatického výskumu, ale aj mnohí matematici ho vidia ako jednu z hlavných a najťažších výskumných úloh matematiky.

5.6 Pomoc, mám ťažký problém...

Volanie o pomoc v nadpise by sme nemali podceňovať. Najlepšie známe algoritmy pre NP-ťažké problémy vyžadujú pre vstupy praktickej veľkosti viac výpočtovej práce, ako je možné v tomto vesmíre vykonať. Nie je to dostatočný dôvod na volanie o pomoc? Najmä keď je riešenie týchto problémov spojené s otázkami bezpečnosti alebo s veľkými finančnými investíciami.

No má volanie o pomoc zmysel? Je niekto, kto by mohol pomôcť pri polynomiálne neriešiteľných úlohách? Áno, existuje záchrana. Pomôcť môžu iba algoritmici. Sú ozajstnými umelcami a divotvorcami pri hľadaní riešenia problémov. Mnohé problémy sú nestabilné (citlivé) v nasledujúcom zmysle. Veľmi malá zmena zadania problému alebo požiadaviek na hľadané riešenie môže zapríčiniť obrovskú zmenu množstva výpočtovej práce, potrebnej na nájdenie riešenia. Zmiernenie (zoslabenie) jedinej podmienky

kladenej na riešenie problému, ktorá je pre prax okrajová, môže znamenať skok z miliardy rokov trvajúceho výpočtu na niekoľko sekundový výpočet. To isté sa môže stať, keď požadujeme namiesto optimálneho riešenia optimalizačného problému len riešenie, ktoré je blízko optimálneho. Blízko optimálneho môže znamenať, že sa odmeraná kvalita vypočítaného riešenia líši od optimálneho najviac o 1%. Takéto algoritmy, ktoré počítajú riešenie blízke optimálnemu, voláme aproximačné algoritmy. Iná možnosť sú randomizované algoritmy, ktoré si volia náhodne stratégiu na hľadanie riešenia. Keď si pre daný prípad problému zvolia náhodou zlú stratégiu, môžu vypočítať nesprávne výsledky alebo výpočet môže trvať veľmi dlho. Keď si vyberú vhodnú stratégiu, vypočítajú rýchlo správne výsledky. V mnohých aplikáciách sme spokojní, že s vysokou pravdepodobnosťou efektívne vypočítame správny výsledok. Aby sme tomu dobre rozumeli, používaním algoritmov využívajúcich náhodu ušetríme výpočtové náklady vďaka zmierneniu (zoslabeniu) požiadavky, že pri každom výpočte vždy vypočítame správny výsledok. Existuje viacero ďalších možností, ktoré sa dajú aj navzájom kombinovať. Je umením algoritmikov objavovať, kde sú slabiny ťažkých problémov. Ich úlohou je zaplatiť čo najmenej (zľaviť z našich požiadaviek tak málo ako sa len dá) za možný skok od nerealizovateľného množstva práce počítača k realizovateľným nákladom. V niektorých prípadoch za tento skok zľavíme z našich požiadaviek tak málo, že sa to zdá až neuveriteľné a dá sa vravieť o skutočnom zázraku. Takýto div náhodného riadenia predstavíme v nasledujúcej kapitole.

Uvedieme iba malý príklad ako efektívne vypočítať namiesto optimálneho len relatívne „dobré“ riešenie. V odseku 5.5 sme predstavili problém pozorovateľov VC ako rozhodovací problém. Jeho optimalizačná verzia MIN-VC hľadá pre cestnú sieť minimálny počet pozorovateľov, ktorí stačia na dohľad nad celou sieťou. Pre sieť na obr. 5.5 sú to dvaja pozorovatelia.

Úloha 5.19 Aký je minimálny počet pozorovateľov na dohľad siete, ktorá je na obr. 5.7? Zdôvodnite svoju odpoveď!

Aby sme boli názorní a nepreťažili nešpecialistov, vysvetlíme len, ako nájdeme efektívne riešenie, ktoré bude v najhoršom prípade vyžadovať dvakrát toľko pozorovateľov ako by bolo nevyhnutne treba. Hoci by sa výsledok mohol zdať ľahko dosiahnuteľný, v sieťach s desaťtisícmi križovatiek človek môže ťažko zaručiť kvalitnejšie ad-hoc riešenie.

Myšlienka hľadať riešenie, ktoré bude v najhoršom prípade vyžadovať

dvojnásobný[9] počet pozorovateľov, je založená na tom, že každá ulica $Ulica(K_1, K_2)$ medzi dvomi križovatkami K_1 a K_2 sa dá pozorovať len z K_1 alebo K_2. Takže aspoň na jednu z križovatiek K_1 alebo K_2 musíme umiestniť pozorovateľa. Z toho sa dá vytvoriť nasledujúca stratégia, ktorá určite nájde prípustné[10] riešenie.

Algoritmus: App-VC

Vstup: Sieť N

Postup:

1. *Zvoľ si ľubovoľnú ulicu $Ulica(K_1, K_2)$. Umiestni pozorovateľov na obidvoch križovatkách K_1 a K_2. Vytvor menšiu cestnú sieť N', v ktorej odstrániš všetky ulice, ktoré sú pod dohľadom pozorovateľov na K_1 a K_2 (odstránime tým všetky ulice vychádzajúce s K_1 a K_2, teda aj $Ulica(K_1, K_2)$).*

2. *Rovnakou stratégou pokračuj so sieťou N'.*

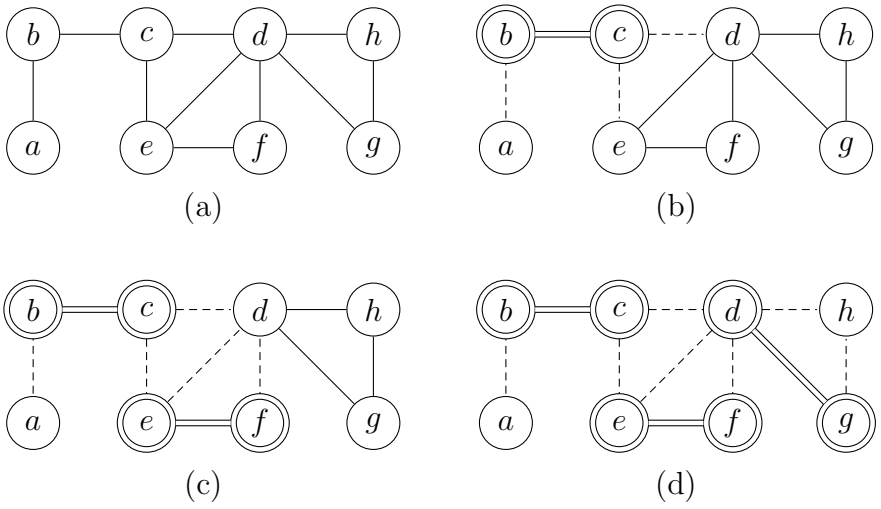
3. *Pokračuj v práci, až kým nie sú zo siete odstránené všetky ulice, čo znamená, že všetky sú pod dohľadom.*

Vykonajme algoritmus App-VC pre sieť na obr. 5.9(a) s 8 križovatkami a, b, c, d, e, f, g a h.

Predpokladajme, že App-VC vyberie ako prvú ulicu $Ulica(b, c)$. Označme na obr. 5.9(b) túto voľbu dvojitou čiarou. Teraz odstránime všetky ulice, ktoré sa končia na križovatkách b alebo c. Na obr. 5.9(b) sú znázornené prerušovanými čiarami. Plné čiary predstavujú 6 ulíc, ktoré ostali v sieti. Predpokladajme, že nasledujúcu ulicu by App-VC vybral $Ulica(e, f)$ (obr. 5.9(c)). Ako vidíme na obr. 5.9(c), umiestnenie pozorovateľov na križovatky e a f, spôsobí odstránenie ulíc $Ulica(e, f)$, $Ulica(e, d)$ a $Ulica(f, d)$. V sieti ostali len tri ulice $Ulica(d, g)$, $Ulica(d, h)$ a $Ulica(h, g)$. Keď si teraz App-VC vyberie ulicu $Ulica(d, g)$, pridáme dvoch pozorovateľov na d a g, a tým sú pod dohľadom aj všetky tri zvyšné ulice. Uvedené použitie App-VC nás priviedlo k rozmiestneniu 6 pozorovateľov na križovatkách b, c, e, f, d a g.

Všimnime si, že výsledky App-VC môžu byť rozličné v závislosti od náhodnej voľby ulíc, ktoré ešte nie sú pod dohľadom.

[9] v porovnaní s optimálnym riešením
[10] Riešenie, ktoré garantuje dohľad nad sieťou.



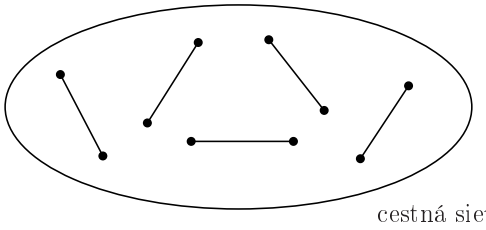
obr. 5.9

Úloha 5.20 Koľko pozorovateľov stačí, aby bola pod dohľadom sieť na obr. 5.9(a)? Nájdite takú „nešťastnú“ voľbu ulíc, že vo výslednom riešení App-VC budú pozorovatelia umiestnení na všetkých 8 križovatkách. Existuje taká voľba ulíc App-VC, ktorej výsledkom bude menej ako 6 pozorovateľov?

Úloha 5.21 Použite algoritmus App-VC na rozmiestnenie pozorovateľov v sieťach na obr. 5.5 a na obr. 5.7.

Nakoniec by sme chceli ešte zdôvodniť, prečo riešenia, ktoré dostaneme použitím App-VC, nie sú oveľa horšie ako optimálne. Všimnime si ulice pod dohľadom pozorovateľov, na obr. 5.9 sme ich označili dvojitými čiarami. Nijaké dve z nich sa nestretávajú na niektorej križovatke. Je to preto, lebo po vybraní ulice $Ulica(K_1, K_2)$ sme odstránili všetky ulice vychádzajúce z K_1 a K_2. Takže žiadna z neskôr vybraných ulíc nemôže viesť do K_1 alebo K_2. Keď si všimneme tieto vybrané ulice v nejakej sieti (obr. 5.10), vidíme, že sú to izolované ulice, ktoré sa nestretávajú na nijakej križovatke (pozri aj obr. 5.9(d)).

Riešenie úlohy pozorovateľov vyžaduje, aby boli všetky ulice pod dohľadom. Keďže vybrané ulice sú súčasťou siete, musia aj ony byť pod dohľadom. Pretože sú izolované, jeden pozorovateľ môže mať dohľad najviac nad jednou z nich. Preto musí mať každé prípustné riešenie k dispozícii aspoň toľko pozorovateľov, koľko je týchto izolovaných ulíc. Riešenie nájdené App-VC umiestni pozorovateľa na všetky konce (križovatky) vy-



obr. 5.10

braných ulíc. Takto dostaneme, že App-VC nikdy nevypočíta riešenie, ktoré má viac ako dvojnásobný počet nevyhnutných pozorovateľov.

Záruka kvality App-VC na nás isto nespravila veľký dojem. Je ale viacero náročnejších príkladov efektívnych algoritmov, ktoré vypočítajú riešenie pre NP-ťažké optimalizačné problémy a kvalita riešenia sa líši od optimálneho menej ako 1‰. A v týchto prípadoch už môžeme vravieť o zázraku algoritmiky.

5.7 Zhrnutie

Popri pojmoch algoritmus a program je pojem výpočtová zložitosť v informatike najdôležitejší a jeho význam preniká čoraz viac do ďalších vedeckých disciplín. Časová zložitosť je najdôležitejšia výpočtová zložitosť, ktorá meria množstvo práce (počet vykonaných výpočtových operácií) algoritmu. Časovú zložitosť algoritmu chápeme ako zvyčajne rastúcu funkciu, ktorej argumentom je veľkosť vstupu.

Hlavná úloha teórie zložitosti je klasifikácia algoritmických problémov vzhľadom na nevyhnutnú a postačujúcu výpočtovú zložitosť na ich riešenie, a tým aj na rozdelenie algoritmicky riešiteľných problémov na prakticky riešiteľné a prakticky neriešiteľné. Je ťažké presne určiť hranicu medzi prakticky riešiteľnými a prakticky neriešiteľnými problémami. V prvom priblížení sa navrhuje táto hranica tak, že problémy, ktoré sa dajú riešiť v polynomiálnom čase, sú prakticky riešiteľné. Problémy, na ktorých riešenie neexistujú polynomiálne algoritmy, považujeme za ťažké (prakticky neriešiteľné). Teória zložitosti nás učí, že existujú ľubovoľne ťažké výpočtové problémy, napríklad také, ktoré sa dajú riešiť iba algoritmami s časovou zložitosťou aspoň 2^{2^n}.

Ústrednou a najťažšou úlohou základného výskumu v informatike je dokazovanie neexistencie efektívnych algoritmov pre riešenie konkrétnych

problémov. Na tento účel nám zatiaľ chýbajú matematické prostriedky, preto sa uspokojíme s neúplnými argumentmi na zdôvodnenie obťažnosti konkrétnych problémov. Metóda redukcie sa zasa využíva, aby sa efektívnou redukciou v polynomiálnom čase ukázalo, že tisíce problémov sú vzhľadom na praktickú riešiteľnosť rovnako ťažké. Nazývame ich NP-ťažké problémy. Pre nijaký NP-ťažký problém nepoznáme efektívny algoritmus; najlepšie algoritmy majú exponenciálnu časovú zložitosť. Existencia polynomiálneho algoritmu pre hociktorý NP-ťažký problém by znamenala, že tisíce problémov, ktoré považujeme za ťažké, by sa dali efektívne riešiť. To je jeden z hlavných dôvodov, prečo veríme, že NP-ťažké problémy sú naozaj ťažké.

Skutočné umenie algoritmika je v hľadaní riešení ťažkých problémov. Najdôležitejší objav je, že mnohé ťažké problémy sú veľmi nestabilné vzhľadom na ich obťažnosť. Malá zmena zadania problému alebo malé zoslabenie požiadaviek môže spôsobiť skok od fyzikálne nerealizovateľného množstva výpočtovej práce k pár sekundám výpočtu na obyčajnom PC. Objavenie a využitie tejto citlivosti ťažkých úloh je jadrom dnešnej algoritmiky s „neoceniteľným“ prínosom pre priemysel a v neposlednom rade aj pre rozvoj informačnej spoločnosti.

Návody riešenia vybraných úloh

Úloha 5.2 Trojnásobným použitím distributívneho zákona môžeme nasledujúcim spôsobom prepísať polynóm 4-tého stupňa:

$$\begin{aligned} & a_4 \cdot x^4 + a_3 \cdot x^3 + a_2 \cdot x^2 + a_1 \cdot x + a_0 \\ = \quad & [a_4 \cdot x^3 + a_3 \cdot x^2 + a_2 \cdot x^1 + a_1] \cdot x + a_0 \\ = \quad & [(a_4 \cdot x^2 + a_3 \cdot x + a_2) \cdot x + a_1] \cdot x + a_0 \\ = \quad & [((a_4 \cdot x + a_3) \cdot x + a_2) \cdot x + a_1] \cdot x + a_0 \quad . \end{aligned}$$

Keď použijeme tento vzorec na výpočet hodnoty polynómu pre zadané čísla a_4, a_3, a_2, a_1, a_0 a x, potrebujeme len štyri násobenia a štyri sčítania.

Úloha 5.4

a) Hodnotu x^6 môžeme vypočítať nasledujúcou stratégiou s 3 násobeniami:

$$I \leftarrow x \cdot x \ , \quad J \leftarrow I \cdot I \ , \quad Z \leftarrow J \cdot I.$$

Vidíme, že postup je založený na myšlienke, najskôr vypočítať jedným násobením $x^2 = x \cdot x$ a potom $x^6 = x^2 \cdot x^2 \cdot x^2$.

b) Hodnotu x^{64} môžeme vypočítať 6 násobeniami nasledujúco:

$$x^{64} = (((((x^2)^2)^2)^2)^2)^2$$

Túto stratégiu môžeme nasledujúcim spôsobom zapísať ako inštrukcie na výpočet

$$\begin{array}{lll} I \leftarrow x \cdot x\ , & I \leftarrow I \cdot I\ , & I \leftarrow I \cdot I\ , \\ I \leftarrow I \cdot I\ , & I \leftarrow I \cdot I\ , & I \leftarrow I \cdot I\ . \end{array}$$

c) Stratégiou výpočtu x^{18} môže byť:

$$x^{18} = (((x^2)^2)^2)^2 \cdot x^2$$

a prepíšeme ju nasledovne:

$$\begin{array}{lll} I \leftarrow x \cdot x\ , & J \leftarrow I \cdot I\ , & J \leftarrow J \cdot J\ , \\ J \leftarrow J \cdot J\ , & Z \leftarrow I \cdot J & . \end{array}$$

d) Hodnotu x^{45} môžeme vypočítať stratégiou:

$$\begin{array}{rcl} x^{45} & = & x^{32} \cdot x^8 \cdot x^4 \cdot x \\ & = & ((((x^2)^2)^2)^2)^2 \cdot (((x^2)^2)^2 \cdot (x^2)^2 \cdot x, \end{array}$$

po prepísaní

$$\begin{array}{lll} I_2 \leftarrow x \cdot x\ , & I_4 \leftarrow I_2 \cdot I_2\ , & I_8 \leftarrow I_4 \cdot I_4\ , \\ I_{16} \leftarrow I_8 \cdot I_8\ , & I_{32} \leftarrow I_{16} \cdot I_{16}\ , & Z \leftarrow I_{32} \cdot I_8\ , \\ Z \leftarrow Z \cdot I_4\ , & Z \leftarrow Z \cdot x & \end{array}$$

s 8 násobeniami.

V prípade, že pre x^{45} zvolíme nasledujúcu špeciálnu na mieru šitú stratégiu:

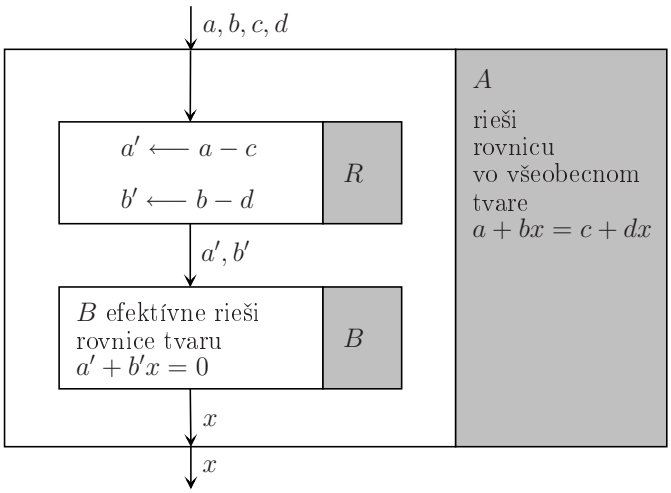
$x^2 = x \cdot x$, $x^3 = x^2 \cdot x$, $x^6 = x^3 \cdot x^3$, $x^9 = x^6 \cdot x^3$,
$x^{18} = x^9 \cdot x^9$, $x^{36} = x^{18} \cdot x^{18}$, $x^{45} = x^{36} \cdot x^9$,

vystačíme len so 7 násobeniami.

Úloha 5.9 Keď má matematik vyriešiť lineárnu rovnicu $a + bx = c + dx$, zjednoduší si ju nasledujúcim spôsobom:

$$\begin{array}{rcll} a + bx & = & c + dx & | - c \\ a - c + bx & = & dx & | - dx \\ (a - c) + bx - dx & = & 0 & \\ (a - c) + (b - d) \cdot x & = & 0 & \{\text{využitím distributívneho zákona}\} \end{array}$$

Teraz je nová rovnica v požadovanom zjednodušenom tvare, pre ktorý už máme algoritmus. Na obr. 5.11 je grafické znázornenie tejto efektívnej redukcie.



obr. 5.11

Redukcia R je efektívna, lebo na to, aby sme upravili všeobecnú lineárnu rovnicu na tvar $a' + b'x = 0$ stačia práve dve odčítania. Algoritmus B zoberie z R čísla a' a b' a vyrieši rovnicu $a' + b'x = 0$. Keďže $a' + b'x = 0$ je len iný tvar rovnice $a + bx = c + dx$, majú obe rovnice rovnaké riešenie, a teda algoritmom B vypočítaná hodnota pre x je aj výstupom algoritmu A pre riešenie lineárnej rovnice vo všeobecnom tvare.

Úloha 5.10 Musia byť aspoň štyria, lebo z K_4 vedú štyri ulice a keď nemôžeme umiestniť pozorovateľa na K_4, musia byť štyria pozorovatelia na opačných koncoch týchto štyroch ulíc. Títo štyria pozorovatelia majú pod dohľadom celú cestnú sieť (obr. 5.5).

Úloha 5.17 Pre 7 križovatiek a najviac 3 pozorovateľov dostaneme nasledujúce lineárne nerovnosti:

$$x_1 + x_2 + x_3 + x_4 + x_5 + x_6 + x_7 \leq 3 \ .$$

Pre 8 ulíc dostaneme nesledujúcich 8 nerovností:

$$x_1 + x_5 \geq 1 \ , \ x_1 + x_6 \geq 1 \ , \ x_2 + x_5 \geq 1 \ , \ x_2 + x_6 \geq 1 \ ,$$

$$x_3 + x_6 \geq 1 \ , \ x_3 + x_7 \geq 1 \ , \ x_4 + x_6 \geq 1 \ , \ x_4 + x_7 \geq 1 \ .$$

Vzhľadom na prvú nerovnosť môžeme priradiť najvac trom neznámym hodnotu 1. Voľba $x_5 = x_6 = x_7 = 1$ a $x_1 = x_2 = x_3 = x_4 = 0$ zaručí splnenie zvyšných 8 nerovností.

Náhoda praje iba pripravenej duši, nepripravená duša nevidí pomocnú ruku, ktorú mu podáva šťastena.

Louis Pasteur

Kapitola 6

Náhoda a jej úloha v prírode alebo náhoda ako zdroj efektívnosti v algoritmike

6.1 Vytýčenie cieľa

V tejto kapitole chceme predstaviť jeden z najväčších divov informatiky. Tento div je založený na neuveriteľne šikovnom využití náhody pri návrhu efektívnych algoritmov pre zdanlivo prakticky neriešiteľné úlohy.

Prv ako tento div priblížime, musíme sa najprv trochu zoznámiť s významom pojmu náhoda. Z tohto dôvodu najprv nahliadneme do histórie vied, v ktorých donekonečna prebiehali spory o existencii skutočnej náhody. Pritom by sme mali nadobudnúť predstavu, čo je pravá náhoda, a čo za pravú náhodu nemôžeme považovať.

So získanými predstavami o význame pojmu náhoda sa vydáme rovno do informatiky, aby sme využili náhodu pri efektívnom riešení ťažko vypočítateľných problémov. Zažijeme pritom zázrak. Uverili by ste, že systém, v ktorom je šikovne použitá náhoda, nás môže priviesť k cieľu miliardukrát rýchlejšie ako akýkoľvek plne deterministický systém? A že toto urýchlenie sa dá dosiahnuť bez toho, aby sme museli podstúpiť väčšie

riziko ako 1 k 10^{19}, že vypočítaný výsledok bude nesprávny[1]. Je toto riziko vážne? Ak by sme od Veľkého tresku každú sekundu spustili výpočet randomizovaného systému (t.j. medzi 10^{18} a 10^{19} krát) môžeme očakávať, že nijaký z toho obrovského počtu výpočtov nebude chybný. Inak povedané pravdepodobnosť aspoň jednej chyby spomedzi viac ako 10^{18} experimentálnych výpočtov randomizovaného systému je menšia ako pravdepodobnosť, že všetky budú správne.

Upozorňujeme, že v praxi môže byť randomizovaný algoritmus (algoritmus využívajúci náhodu) s veľmi malou pravdepodobnosťou chyby spoľahlivejší ako jeho najlepší deterministický náprotivok. Čo tým myslíme? Teoreticky sa deterministické algoritmy nesmú mýliť. Ale v praxi nie sú deterministické programy absolútne spoľahlivé pretože počas ich výpočtu sa môžu vyskytnúť chyby hardvéru, ktoré spôsobia chybné výsledky. Pravdepodobnosť výskytu hardvérovej chyby rastie úmerne s časom výpočtu programu. Preto rýchly randomizovaný algoritmus môže byť spoľahlivejší ako pomalý deterministický program. Napríklad randomizovaný algoritmus, ktorý vypočíta výsledok za 10 sekúnd s pravdepodobnosťou chyby $1/10^{30}$ je spoľahlivejší ako deterministický program, ktorý vypočíta výsledok za týždeň. Takže posun od bežných (deterministických) algoritmov a systémov k randomizovaným nemusí znamenať zníženie spoľahlivosti. Strata zdanlivej absolútnej spoľahlivosti nie je až tak bolestivá. Randomizácia, myslíme tým vytváranie systémov šikovným kombinovaním deterministických stratégií a náhodných rozhodnutí, sa stáva novou paradigmou informatiky. Efektívne randomizované algoritmy, ktoré robia chybu v priemere v jednom z miliárd použití, sú praktické a umožňujú riešenie problémov, ktoré považujeme za neriešiteľné deterministickými algoritmami.

Na konci kapitoly by ste nemali len dôvernejšie poznať konkrétny náhodný systém a jeho výhody, ale budete mať možnosť sa aj hlbšie zoznámiť s myšlienkami, na ktorých je založená randomizácia (použitie náhody v algoritmoch) a získať pritom čiastočne aj intuíciu, prečo vôbec existujú takéto neuveriteľné a zázračné spôsoby riešenia problémov. Na záver sa vás trochu pokúsim povzbudiť, aby ste tieto vedomosti využili v každodennom živote a s ich pomocou sami vytvárali ďalšie „zázraky“.

[1] Riziko vypočítania chybného výsledku nazývame pravdepodobnosť chyby.

6.2 Čo je náhoda a existuje vôbec ozajstná náhoda?

V prvej kapitole sme pochopili, že pre vytváranie a rozvoj vedeckých disciplín je základom tvorba pojmov. Pojem **náhoda** je jedným z mála fundamentálnych a zároveň jedným z najviac diskutovaných vedeckých pojmov. Filozofi, fyzici, biológovia a chemici už tisícročia diskutujú, či existuje náhoda a skúmajú jej úlohu vo svete. Matematici vyvinuli teóriu pravdepodobnosti ako nástroj na skúmanie náhodných javov a aplikujú ju v štatistike. Informatika skúma, či, prečo a kedy je pre nás výhodné využiť náhodu. Inžinierske disciplíny aplikujú získané poznatky pri vývoji efektívnych a spoľahlivých systémov využívajúcich náhodu.

Pojem náhody je úzko spätý s ďalšími dvomi základnými pojmami: **determinizmom** (určenosťou) a **indeterminizmom**[2] (neurčenosťou). Deterministická predstava sveta je založená na princípe kauzality. Každá udalosť má svoju príčinu. Príčina má následok a ten je zase príčinou pre ďalší následok, atď. Podľa takéhoto chápania, ak poznáme všetky zákonitosti a momentálny stav vesmíru, vieme presne predpovedať, čo sa stane v budúcnosti. Na celé dejiny sa môžeme pozerať ako na reťaz príčin a ich následkov. Nedeterminovanosť znamená, že príčina môže mať viacero rôznych následkov a neexistuje prírodný zákon, ktorý by určoval, ktorý z nich nastane.

Pojem náhody je úzko spätý s indeterminizmom. Podľa definície v slovníku

> *jav označujeme ako* ***náhodný****,*

keď

> *sa vyskytuje nepredvídateľne.*

Podobne

> *objekt považujeme za* ***náhodný****,*

keď

> *vznikol alebo bol vytvorený bez akéhokoľvek plánu a vzoru.*

Prirodzene je existencia náhodných udalostí (teda výskytu náhody) v protiklade s deteministicko–príčinnou predstavou o fungovaní nášho sveta. Z tohto dôvodu je základná otázka vedy,

[2] Informatici používajú termín nedeterminizmus.

> *či náhoda existuje objektívne, alebo len používame tento pojem, aby sme vysvetlili a modelovali javy, ktorých zákonitosti nepoznáme.*

O odpovedi na túto otázku vedci vedú spory už od antiky. *Demokritos* sa nazdával a tvrdil, že

> *nahodné je nepoznané a že príroda je vo svojej podstate deterministická.*

Tým myslel, že svet sa riadi pevným poriadkom, ktorý určujú jednoznačné zákony. Podľa Demokrita pojem náhoda používame iba subjektívne, aby sme zastreli naše nepochopenie vecí a javov. Takto by bol pojem náhodnosti iba dôsledkom neúplnosti našich vedomostí. Svoj postoj Demokritos rád ilustroval nasledujúcim príkladom.

Dvaja páni pošlú svojich otrokov zámerne v tom istom čase k studni po vodu. Otroci sa pri studni stretnú a povedia si: „To je ale náhoda, že sme sa tu stretli.“ Ale v skutočnosti bolo ich stretnutie určené rozhodnutím ich pánov, ale keďže o tom nevedeli, javilo sa otrokom ako náhodná udalosť.

Epikuros oponoval Demokritovi nasledujúcim tvrdením:

> *„Náhoda je objektívna, je prirodzenou vlastnosťou javov.“*

Epikuros tým myslel, že objektívna náhoda existuje nezávisle od našej vedomosti alebo nevedomosti. Podľa Epikura existujú procesy, ktorých priebeh nie je jednoznačný, ale viacznačný a nepredvídateľný a voľbu spomedzi existujúcich možností nazývame náhoda.

Niekto by mohol povedať: „Je zrejmé, že pravdu má Epikuros, veď v hazardných hrách ako kocky alebo ruleta môžu byť rôzne výsledky a len náhoda určí, čo bude výsledok.“ No vec nie je taká jednoduchá a úvahy o kockách a rulete podporujú skôr mienku Demokrita. Hra s kockami je veľmi komplexný jav, ale keby sme vedeli, akou rýchlosťou, ktorým smerom a na aký povrch ich hodíme, dal by sa dopredu určiť výsledok - ktoré číslo padne na kocke. Prirodzene je pohyb ľudskej ruky v spojení s mozgom príliš zložitý na to, aby sme vedeli určiť všetky parametre potrebné na vypočítanie výsledku. Tento proces ale nemôžeme chápať ako objektívne náhodný, len na základe toho, že je priveľmi zložitý. Podobne je to aj s hádzaním guličky v rulete a iných hrách založených na náhode. Aj vo fyzike sa na mnohých miestach používa na opis a skúmanie fyzikálnych procesov pravdepodobnostný model, hoci nejde o jednoznačne náhodné procesy (a niekedy sú dokonca očividne deterministické). Tieto

sú iba priveľmi zložité na to, aby sme ich modelovali plne deterministickým spôsobom. Je zaujímavé, že z tohto istého dôvodu pripúšťal aj Albert Einstein použitie pojmu náhoda len pri modelovaní niečoho ešte nie úplne poznaného a veril[3], že existujú jednoduchšie a jasnejšie formulovateľné deterministické prírodné zákony.

Do 20. storočia sa akceptovalo prevažne Demokritovo kauzálno-deterministické poňatie. Dôvody boli na jednej strane v náboženstve, kde sa v Bohom stvorenom svete nepripúšťala existencia náhody[4] a na druhej strane vo veľkom pokroku prírodných vied a techniky dosiahnutom v 19. storočí, čo spôsobilo prílišný optimizmus, že sa všetko dá objaviť a všetko objaviteľné sa dá formulovať priamočiaro sledom príčin a dôsledkov.

Netreba podceniť ani úlohu emócií v vzťahu k náhode[5]. Možnosť existencie náhody sa vtedy pociťovala veľmi negatívne, lebo náhoda sa považovala za synonymum neželateľnej neistoty, chaosu a nepredvídateľnosti. Negatívny postoj k existencii náhody asi najlepšie sformuloval vo svojich meditáciách Marcus Aurelius:

> *„Sú len dve možnosti: buď vo svete vládne obrovský chaos alebo poriadok a zákon.“*

Je zrejmé, že aj vedci, ktorí si osvojili názor, že náhoda je emocionálne neželateľná, sa snažili vo výskume vystačiť bez akéhokoľvek uvažovania o potenciálnej existencii náhody. Prípadne išli ešte o krok ďalej a snažili sa výsledkami svojej práce do popredia stavať deterministickú príčinnosť a úplne vylúčiť existenciu náhody.

Až s rozvojom vedy v 20. storočí sme rozpoznali, že bližšie k realite môže byť Epikurovo chápanie náhody. Matematické modely evolúcie ukazujú, že by sa neuskutočnila bez náhodných mutácií (náhodných zmien v DNA sekvenciách). K uznaniu existencie náhody prispeli podstatnou mierou úspechy fyziky, predovšetkým kvantovej mechaniky. Tento matematický model správania sa častíc mikrosveta je založený na viacznačnosti, ktorú opisujeme náhodnými udalosťami. Experimentálne sa potvrdili všetky dôležité predpovede založené na tejto teórii, preto určité udalosti v mikro-

[3] „Boh nehádže kockami“ je jeden z najznámejších výrokov Alberta Einsteina. A rovnako známa je odpoveď Nielsa Bohra „Boh si nenechá hovoriť do toho, čo má robiť.“

[4] Dnes už vieme, že takáto interpretácia nie je správna a ozajstná náhoda nie je v protiklade s existenciou Boha.

[5] O postavení emócií vo výskume hovoria takzvané exaktné vedy nerady, ale je to len popieranie skutočnosti, že aj vo vede môžu byť emócie na jednej strane hnacím motorom rozvoja a pokroku, a na druhej strane môžu výskum spomaľovať alebo poznanie aj na celé roky „zakonzervovať“.

svete interpretujeme ako naozajstné náhody.[6] Uznanie existencie náhodných javov (náhody) je ale dôležité aj preto, lebo ich modelovaním sa prepája náhoda s neurčitosťou a chaosom. Najvýstižnejšie to sformuloval maďarský matematik Alfréd Rényi:

> *„Neexistuje protirečenie medzi príčinnosťou a náhodou. Vo svete vládne náhoda, a práve preto vládnu vo svete poriadok a zákonitosti prejavujúce sa prostredníctvom množstva náhodných udalostí, ktoré sa správajú podľa zákonov teórie pravdepodobnosti.“*

My informatici máme aj iný dôvod zaoberať sa náhodou, ako je modelovanie prírodných procesov a empirický štatistický výskum. Najlepšie to vyjadril prekvapujúco už pred 200 rokmi veľký básnik Johann Wolfgang von Goethe:

> *„Tento svet je vytvorený spleťou nevyhnutnosti a náhody; Ľudský rozum sa stavia medzi ne a vie ich obe ovládať. Nevyhnutné považuje sa základ bytia; náhodné vie usmerniť, viesť a využiť.“*

V tomto zmysle je Johann Wolfgang von Goethe „prvým informatikom“, ktorý vidí v náhode (randomizácii) užitočný nástroj na realizáciu niektorých činností. Táto kapitola sa venuje použitiu náhody ako zdroju nevídanej výpočtovej efektívnosti. Naším cieľom bude ukázať, že namiesto plne deterministických systémov a algoritmov sa často oplatí navrhnúť a implementovať (randomizovaný) systém riadený náhodou. Nie je to nič iné ako akceptovanie prírody ako učiteľa. Ukazuje sa, že príroda na dosiahnutie cieľa vždy používa najefektívnejší a najjednoduchší spôsob a využitie náhody je podstatnou črtou jej stratégie. Informatická prax tento prístup potvrdzuje. V mnohých dnešných aplikáciách pracujú jednoduché randomizované algoritmy s vysokou mierou spoľahlivosti, pričom nepoznáme nijaké deterministické algoritmy, ktoré by robili to isté porovnateľne efektívne. Dokonca máme príklady randomizovaných algoritmov, pre ktoré je návrh a použitie ich deterministických náprotivkov fyzikálne neuskutočniteľné. Z týchto dôvodov v súčasnosti nespájame praktickú riešiteľnosť s efektívnosťou deterministických algoritmov, ale s efektívnosťou randomizovaných algoritmov.

Aby sme presvedčili aj informaticky nepodkutého čitateľa o neuveriteľnej sile randomizácie na jednoduchom a názornom príklade, ukážeme v nasledujúcej časti randomizovaný komunikačný protokol, ktorým vyriešime

[6] Túto tému podrobnejšie preberieme v kapitole o kvantových počítačoch.

zadanú úlohu s podstatne menšou komunikáciou, ako je nevyhnutne potrebná pri najlepšom možnom deterministickom komunikačnom protokole.

Dôležitá poznámka na záver: Nedali sme odpoveď na otázku, či existuje skutočná náhoda a je veľmi nepravdepodobné, že by veda na túto otázku v najbližšom čase našla definitívnu odpoveď. Je to jednoducho preto, lebo je to viac fundamentálna otázka v rovine vedeckých axióm, ako v rovine výsledkov výskumu. A ako sme sa dozvedeli už v prvej kapitole, na tejto základnej axiomatickej úrovni nemajú ani exaktné vedy matematika alebo fyzika všeobecne platné tvrdenia, ale len ako axiómy vyslovené domnienky zodpovedajúce všetkým doterajším skúsenostiam (ako sme už spomínali v kapitole 1).

Ako informatici si dovoľujeme veriť v existenciu náhody nielen preto, lebo tento názor podporujú doterajšie skúsenosti fyziky a evolučnej teórie, ale predovšetkým preto, lebo náhoda je zdrojom efektívnosti. Náhoda nám umožňuje určité ciele dosiahnuť miliardukrát rýchlejšie a informatik by bol veľmi prekvapený, keby príroda nechala nepovšimnutú takúto možnosť urýchlenia procesov.

6.3 Mnoho svedkov je pomoc v núdzi alebo prečo môže byť náhoda užitočná?

Na príklade jednoduchej úlohy si ukážeme, ako sa táto dá veľmi efektívne vyriešiť využitím náhody, hoci vieme, že deterministicky (bez náhody) sa efektívne vyriešiť nedá. Bude to príklad, ktorým si ozaj každý môže samostatne vyskúšať silu použitia náhody.

Čo presne znamená, keď vravíme o systéme alebo algoritme (programe), že **využíva náhodu** (je **randomizovaný**)? V zásade sú možné dve predstavy. Každý deterministický program vykoná pre ľubovoľné vstupné údaje tú istú postupnosť výpočtových krokov. Hovoríme, že výpočet je jednoznačný. Randomizovaný program (systém) môže mať pre jeden vstup viacero rozličných výpočtov. Je vecou náhody, ktorý z nich sa ozaj uskutoční. Uvedieme dva spôsoby voľby z viacerých možných výpočtov.

1. Program pracuje deterministicky s výnimkou niekoľkých miest, kde si môže hodiť mincou. V závislosti od toho, čo padne (rub alebo líc), sa na tomto mieste rozhodne, ktorým z dvoch možných pokračovaní bude výpočet prebiehať.

Náš programovací jazyk týmto spôsobom môžeme názorne rozšíriť pridaním nasledujúcej inštrukcie:

```
Hoď mincou. V prípade, že padne „líc“ pokračuj
v riadku i, inak pokračuj v riadku j
```

Takže keď padne „líc (hlava)“, program pokračuje vykonávaním príkazov v riadku i, keď padne „rub (znak)“ pokračuje v riadku j.

2. Program využívajúci náhodu má na začiatku možnosť voľby z viacerých deterministických stratégií. Program si náhodne vyberie jednu z nich a použije ju pre aktuálne vstupné údaje. Zvyšok výpočtu je úplne deterministický. Pri ďalšom vstupe sa náhodne vyberie nová stratégia.

Druhá možnosť modelovania systémov využívajúcich náhodu je jednoduchšia, preto ju použijeme aj v našom vzorovom príklade.

Zamyslime sa nad nasledujúcou úlohou:

Majme dva počítače R_I a R_{II} spojené sieťou, ktoré sú na dvoch od seba vzdialených miestach. Oba počítače majú uchovávať rovnakú databázu. Pôvodne pracovali oba počítače s rovnakými databázami. Postupom času ich obsah dynamicky meníme takým spôsobom, že súčasne vykonávame zmeny v oboch databázach a usilujeme sa, aby na oboch miestach boli rovnaké všetky informácie o skúmaných objektoch (napr. o DNA sekvenciách). Prirodzene čas od času chceme aj preveriť, či sú databázy v R_I a v R_{II} naozaj rovnaké.

Toto preskúšanie si teraz zidealizujeme v zmysle matematickej abstrakcie. Obsahy databáz budeme považovať za postupnosti bitov. Takže v pamäti počítača R_I je postupnosť x dĺžky n bitov

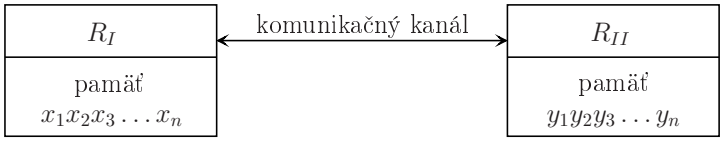
$$x = x_1 x_2 x_3 \dots x_{n-1} x_n,$$

v pamäti počítača R_{II} je tiež postupnosť n bitov, označme ich ako

$$y = y_1 y_2 y_3 \dots y_{n-1} y_n.$$

Úlohou je s využitím komunikácie cez sieť preveriť, či $x = y$ (obr. 6.1).

Aby sme vyriešili túto komunikačnú úlohu, mali by sme navrhnúť stratégiu komunikácie (**komunikačný protokol**). **Zložitosť komunikácie**, a tým aj zložitosť riešenia meriame počtom bitov, ktoré si cez komunikačný kanál vymenia R_I a R_{II}.



obr. 6.1

Žiaľ, dá sa dokázať, že ľubovoľný komunikačný protokol na riešenie tejto úlohy sa nevyhne výmene n bitov. Takže naivné riešenie, pri ktorom R_I pošle celý obsah svojej pamäti $x_1x_2\ldots x_n$ počítaču R_{II} a R_{II} ho bit po bite porovná s obsahom svojej pamäti, je vlastne najlepšou možnou stratégiou. Keď je n veľmi veľké, napr. $n = 10^{16}$ (čo zodpovedá asi 250000 DVD) sú komunikačné náklady príliš vysoké. A to neberieme do úvahy, že spoľahlivé prenesenie takéhoto množstva údajov je nerealistické uskutočniť bez straty alebo zmeny jediného bitu.

Naším cieľom je navrhnúť komunikačný protokol využívajúci náhodu, ktorým sa táto úloha bude dať realizovať

$$4 \cdot \lceil \log_2(n) \rceil$$

komunikačnými bitmi[7]. Vidíme, že ušetríme exponenciálne na zložitosti. Pre $n = 10^{16}$ pošleme napríklad len 256 namiesto 10^{16} bitov!

Kvôli názornosti budeme namiesto porovnávania dvoch postupností bitov porovnávať dve prirodzené čísla. Preto budeme interpretovať postupnosti $x = x_1 \ldots x_n$ a $y = y_1 \ldots y_n$ ako binárne zápisy čísiel:

$$\textbf{Číslo}(\boldsymbol{x}) \quad = \quad \sum_{i=1}^{n} 2^{n-i} \cdot x_i$$

a

$$\textbf{Číslo}(\boldsymbol{y}) \quad = \quad \sum_{i=1}^{n} 2^{n-i} \cdot y_i.$$

V prípade, že vám tieto vzorce nič nevravia, nemusíte sa nimi zapodievať. Dôležité je len vedieť, že

> Číslo(x) je prirodzené číslo, ktorého binárnym zápisom je x a že
>
> $$0 \leq \text{Číslo}(x) \leq 2^n - 1.$$

[7] Pre reálne číslo r, označujeme $\lceil r \rceil$ jeho hornú celú časť, čo je zaokrúhlenie reálneho čísla r na najbližšie väčšie celé číslo. Napríklad pre $r = 3,14$, $\lceil 3,14 \rceil = 4$.

Analogicky platí, že

$$0 \leq \text{Číslo}(y) \leq 2^n - 1.$$

Prirodzene, že pre $n = 10^{16}$ môžu byť tieto čísla obrovské. Už pre $n = 10^6 = 1\,000\,000$ sú približne

$$2^{1\,000\,000}$$

a majú asi 300 000 cifier v desiatkovej číselnej sústave .

Je zrejmé, že x je identické s y práve vtedy, keď Číslo$(x) =$ Číslo(y). Takže sa môžeme sústrediť na porovnávanie čísiel Číslo(x) a Číslo(y) v našom randomizovanom protokole.

Úloha 6.1 Pre tých, čo to chcú pochopiť detailnejšie: Postupnosť 10110 predstavuje číslo:

$$\begin{aligned} \text{Číslo}(10110) &= 1 \cdot 2^4 + 0 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0 \\ &= 1 \cdot 16 + 0 \cdot 8 + 1 \cdot 4 + 1 \cdot 2 + 0 \cdot 1 \\ &= 16 + 4 + 2 = 22. \end{aligned}$$

Ktoré číslo je zakódované postupnosťou bitov 101001? Aká je binárna reprezentácia čísla 133?

Komunikačný protokol využívajúci náhodu bude pre vstupy $x_1 \ldots x_n$ a $y_1 \ldots y_n$ dĺžky n, zodpovedať náhodnému výberu jednej deterministickej stratégie (protokolu) z väčšieho počtu možností. Počet možných stratégií sa bude rovnať počtu prvočísiel menších ako n^2.

V ďalšom pre ľubovoľné kladné celé číslo m označujeme:

$$\text{PRIM}(m) = \{p \text{ je prvočíslo} \mid p \leq m\}$$

množinu všetkých prvočísiel v intervale od 1 po m a

$$\text{Prvoč}(m) = |\text{PRIM}(m)|$$

označuje počet prvočísiel v PRIM(m).

Zvyšok po celočíselnom delení $a : b$ označujeme

$$r = a \bmod b.$$

Napríklad platí, že $2 = 14 \bmod 3$, lebo $14 : 3 = 4$ a zvyšok je $r = 14 - 3 \cdot 4 = 2$.

Keď chápeme delenie $a : b$ ako celočíselné a výsledok je c a zvyšok r, potom môžeme písať:

$$a = b \cdot c + r,$$

kde $r < b$. V našom príklade sú $a = 14$ a $b = 3$, celočíselný podiel je $c = 4$ a zvyšok je $r = 2$. Teda môžeme písať:

$$14 = 3 \cdot 4 + 2,$$

pričom $r = 2 < 3 = b$.

Teraz môžeme nasledujúcim spôsobom opísať protokol využívajúci náhodu na porovnanie x a y (lepšie povedané na porovnanie čísiel Číslo(x) a Číslo(y)).

SVEDOK - Komunikačný protokol využívajúci náhodu na dôkaz zhodnosti.

Východisková situácia: Počítač R_I má n bitov $x = x_1x_2 \dots x_n$ (to znamená číslo Číslo(x), $0 \leq$ Číslo(x) $\leq 2^n - 1$).
Počítač R_{II} má n bitov $y = y_1y_2 \dots y_n$ (teda číslo Číslo(y), $0 \leq$ Číslo(y) $\leq 2^n - 1$).

1. fáza: R_I si náhodne zvolí prvočíslo p z PRIM(n^2). Každé prvočíslo v PRIM(n^2) má rovnakú pravdepodobnosť, že bude zvolené.

2. fáza: R_I vypočíta číslo:

$$s = \text{ Číslo}(x) \bmod p$$

(teda zvyšok po delení čísla Číslo(x) číslom p) a pošle R_{II} binárnu reprezentáciu:

$$s \text{ a } p.$$

3. fáza: Po obdržaní s a p počítač R_{II} vypočíta číslo:

$$q = \text{ Číslo}(y) \bmod p.$$

V prípade, že $q \neq s$, vráti R_{II} odpoveď „sú rozličné“.
V prípade, že $q = s$, vráti R_{II} odpoveď „sú rovnaké“.

Prv ako budeme skúmať komunikačnú zložitosť a spoľahlivosť protokolu SVEDOK, ilustrujeme si na konkrétnom príklade, ako pracuje.

Príklad 6.1 Nech sú $x = 01111$ a $y = 10110$.
Takže

$$\begin{aligned} \text{Číslo}(x) &= 1 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 = 8 + 4 + 2 + 1 = 15, \\ \text{Číslo}(y) &= 1 \cdot 2^4 + 1 \cdot 2^2 + 1 \cdot 2^1 = 16 + 4 + 2 = 22 \end{aligned}$$

a

$$n = 5.$$

Teda máme $n^2 = 25$ a dostaneme

$$\text{PRIM}(n^2) = \{2, 3, 5, 7, 11, 13, 17, 19, 23\}.$$

Preto si komunikačný protokol SVEDOK volí jedno z 9 prvočísiel, a teda aj jeden z 9 deterministických protokolov.
Predpokladajme, že R_I si zvolí prvočíslo 5. Potom ďalej počíta:

$$s = 15 \bmod 5 = 0$$

a pošle čísla $p = 5$ a $s = 0$ počítaču R_{II}. Počítač R_{II} počíta:

$$q = 22 \bmod 5 = 2.$$

Pretože $2 = q \neq s = 0$, odpovedá R_{II} správne, že

„x a y sú rozličné“.

Predpokladajme, že si R_I z PRIM(25) vyberie prvočíslo 7. Vtedy R_I spočíta:

$$s = 15 \bmod 7 = 1$$

a pošle počítaču R_{II} čísla $p = 7$ a $s = 1$. Počítač R_{II} počíta:

$$q = 22 \bmod 7 = 1.$$

Pretože $q = s = 1$, R_{II} odpovie nesprávne:

„x a y sú rovnaké“.

Vidíme, že SVEDOK sa môže mýliť a pre niektoré voľby prvočísla odpovie nesprávne. □

Úloha 6.2 Je ešte iné prvočíslo z PRIM(25), ktoré pre $x = 01111$ a $y = 10110$ spôsobí, že SVEDOK odpovie nesprávne?

Úloha 6.3 Nech sú vstupy $x = y = 100110$. Môže sa stať, že SVEDOK odpovie nesprávne, že „$x \neq y$“ z dôvodu, že si zvolil nevhodné prvočíslo?

Úloha 6.4 Uvažujte, že vstupom sú $x = 10011011$ a $y = 01010101$. Určte presne, koľko prvočísiel spôsobí, že SVEDOK odpovie nesprávne „$x = y$“, a koľko prvočísiel spôsobí, že SVEDOK odpovie správne „$x \neq y$“!

Pozrime sa najprv na zložitosť komunikácie protokolu SVEDOK. Každé z čísiel Číslo(x) a Číslo(y), ktoré porovnávame, je reprezentované n bitmi, nachádzajú sa teda v intervale od 0 po $2^n - 1$. Aby sme sa vyhli posielaniu veľkých čísiel, posiela počítač R_I v protokole SVEDOK len dve čísla menšie ako n^2, konkrétne prvočíslo $p \in PRIM(n^2)$ a zvyšok po delení číslom p. Číslo menšie ako n^2 sa dá zapísať binárne pomocou

$$\lceil \log_2 n^2 \rceil \leq 2 \cdot \lceil \log_2 n \rceil$$

bitov.

Pretože protokol SVEDOK posiela dve čísla p a s a každé sa dá zapísať presne[8] $2\lceil \log_2 n \rceil$ bitmi, teda celkovo sa v protokole SVEDOK pošle presne:

$$4 \cdot \lceil \log_2 n \rceil$$

bitov.

Pozrime sa, čo presne to znamená pre $n = 10^{16}$. Najlepší deterministický protokol sa nevyhne výmene aspoň

$$10^{16} \text{ bitov.}$$

Nášmu randomizovanému protokolu SVEDOK stačí vždy

$$4 \cdot \lceil \log_2(10^{16}) \rceil \leq 4 \cdot 16 \cdot \lceil \log_2 10 \rceil = 256 \text{ bitov.}$$

Rozdiel vo vynaloženej námahe na poslanie 256 bitov a 10^{16} bitov je obrovský. Aj keby bolo bezpečné prenesenie 10^{16} bitov uskutočniteľné, náklady na prenos 256 bitov a 10^{16} bitov sú neporovnateľné. Za túto neuveriteľnú úsporu v komunikačnej zložitosti zaplatíme stratou istoty, že vždy dostaneme správnu odpoveď porovnania. Teraz si kladieme otázku:

> *Aká veľká je miera nespoľahlivosti, ktorou platíme za drastické zníženie komunikačných nákladov?*

Stupeň neistoty, že vypočítaný výsledok je správny, nazývame v informatike **pravdepodobnosť chyby**. Presnejšie, **pravdepodobnosť chyby pre dva vstupné reťazce x a y** je pravdepodobnosť:

$$\mathbf{Chyba}_{SVEDOK}(x, y),$$

[8] V prípade, že by bol zápis kratší ako $2\lceil \log_2 n \rceil$, môžeme ho doplniť na začiatku zopár nulami tak, aby mal presne túto dĺžku.

že SVEDOK odpovie nesprávne na vstup x a y (keď je obsah pamäti R_I reťazec x a obsah pamäti R_{II} reťazec y). Pre rozličné vstupy (počiatočné situácie) x a y môže byť $\text{Chyba}_{SVEDOK}(x, y)$ rôzna. Našou úlohou je ukázať, že pravdepodobnosť chyby je veľmi malá pre všetky možné vstupy x a y.[9]

> *„Aká je presne pravdepodobnosť chyby pre x a y a ako sa dá určiť?“*

Protokol SVEDOK si pre vstupné reťazce x a y dĺžky n náhodne zvolí prvočíslo z $\text{PRIM}(n^2)$. Táto voľba prvočísla rozhodne, či protokol dá správnu alebo nesprávnu odpoveď. Preto si rozdelíme množinu prvočísiel $\text{PRIM}(n^2)$ na dve časti. Všetky prvočísla, ktoré v protokole SVEDOK dajú pre x a y správnu odpoveď, nazveme:

dobré pre (x, y).

V príklade 6.1 bolo prvočíslo 5 dobré pre $(01111, 10110)$.

Ostatné prvočísla, ktorých voľba vedie pre (x, y) k nesprávnemu výsledku nazývame:

zlé pre (x, y).

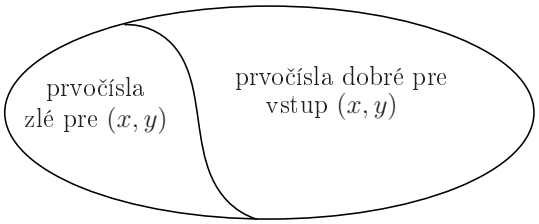
Ako sme videli v príklade 6.1, je prvočíslo 7 zlé pre $(01111, 10110)$. Pretože pre každé z $\text{Prvoč}(n^2)$ prvočísiel z $\text{PRIM}(n^2)$ je rovnako pravdepodobné, že bude zvolené, platí:

$$\text{Chyba}_{SVEDOK}(x, y) = \frac{\text{počet zlých prvočísiel pre } (x, y)}{\text{Prvoč}(n^2)},$$

teda pravdepodobnosť chyby je pomer počtu zlých prvočísiel v urne k počtu všetkých prvočísiel v urne. To je jednoduchá úvaha, ktorú si môžeme názorne ukázať na príklade. Keď máme v urne 15 guličiek, z ktorých sú práve tri biele, potom je pravdepodobnosť toho, že si vytiahneme bielu guličku presne $\frac{3}{15} = \frac{1}{5}$. Inak povedané, 20% všetkých guličiek v urne je bielych, a preto môžeme očakávať, že pri opakovaných ťahoch vytiahneme bielu guličku v 20% prípadov. Analogicky, keď má SVEDOK v $\text{PRIM}(7^2)$ 15 prvočísiel a dve z nich vedú pre x a y k nesprávnej odpovedi, je pravdepodobnosť chyby 2/15.

[9] V tomto smere kladieme na (randomizované) systémy a algoritmy využívajúce náhodu veľmi vysoké požiadavky. Pre každý jeden vstup požadujeme vysokú pravdepodobnosť toho, aby bol výsledok správny. To je v protiklade s takzvanými pravdepodobnostnými postupmi, v ktorých požadujeme len to, že správny výsledok dostaneme s veľkou pravdepodobnosťou na štatisticky významnej podmnožine prípadov problému.

Na obrázku 6.2 znázorňujeme opísanú situáciu. Našou úlohou je ukázať pre všetky páry postupností bitov x a y, že počet zlých prvočísiel pre x a y je výrazne menší ako počet všetkých prvočísiel v množine PRIM(n^2).



obr. 6.2

Aká veľká je množina PRIM(m)? Jedným z najväčších a najdôležitejších objavov matematiky[10] je veta o prvočíslach, ktorá hovorí, že

$$\text{Prvoč}(m) \text{ je približne } \frac{m}{\ln m},$$

kde $\ln m$ je prirodzený logaritmus čísla m. Veta o prvočíslach tvrdí: prvočísla sú rozptýlené medzi prirodzenými číslami relatívne husto, pretože približne každé $\ln m$-té číslo je prvočíslo. Pre naše výpočty si vezmeme konkrétny výsledok, ktorý platí pre všetky prirodzené čísla $m > 67$,

$$\text{Prvoč}(m) > \frac{m}{\ln m}.$$

Pre naše účely sa nám bude hodiť aj, že pre všetky $n \geq 9$ platí

$$\text{Prvoč}(n^2) > \frac{n^2}{2 \ln n}.$$

Náš nasledujúci cieľ je ukázať, že

> *pre ľubovoľný vstup* (x, y) *je počet zlých prvočísiel pre* (x, y) *nanajvýš* $n - 1$,

teda podstatne menej, ako $n^2/(2 \ln n)$.

Pri skúmaní, aká je pravdepodobnosť chyby, budeme rozlišovať dva prípady v závislosti od vzťahu medzi x a y.

1. prípad $x = y$, keď je správna odpoveď „rovnaké“.

V prípade, že $x = y$, platí aj, že Číslo(x) = Číslo(y). Bez ohľadu na to, aké prvočíslo zvolíme, platí:

$$s = \text{ Číslo}(x) \bmod p = \text{ Číslo}(y) \bmod p = q,$$

[10] Presnejšie teórie čísiel v matematike.

a teda $s = q$. Vyjadrené v prirodzenom jazyku, keď prvočíslom p delíme dve rovnaké čísla, musia byť aj oba zvyšky po delení rovnaké. Protokol SVEDOK odpovie správne pre všetky prvočísla p z PRIM(n^2). Preto platí pre všetky reťazce x, že

$$\text{Chyba}_{SVEDOK}(x, x) = 0.$$

Takže chyba sa môže vyskytnúť len v prípade rozličných vstupných reťazcov x a y.

2. prípad $x \neq y$, keď je správna odpoveď „rozličné".

Ako sme už určili pre $x = 01111$ a $y = 10110$ v príklade 6.1, je pravdepodobnosť chyby nenulová, lebo pre $p = 7$ protokol odpovie nesprávne „rovnaké".

Všeobecný horný odhad $n - 1$ počtu zlých prvočísiel pre (x, y) takých, že $|x| = |y| = n$, nájdeme tak, že budeme skúmať vlastnosti týchto zlých prvočísiel.

Prvočíslo p je pre (x, y) zlé, keď sú rovnaké zvyšky po delení čísiel Číslo(x) a Číslo(y) prvočíslom p, to znamená keď platí:

$$s = \text{ Číslo}(x) \bmod p = \text{ Číslo}(y) \bmod p.$$

Rovnosť

$$s = \text{ Číslo}(x) \bmod p$$

neznamená nič iné, len že:

$$\text{Číslo}(x) = h_x \cdot p + s,$$

kde h_x je výsledok delenia čísla Číslo(x) prvočíslom p a $s < p$ je zvyšok po delení.

Analogicky môžeme písať:

$$\text{Číslo}(y) = h_y \cdot p + s,$$

kde sa p nachádza h_y krát v Číslo(y) a s je zvyšok.[11] Predpokladajme, že platí Číslo(x) $\geq$ Číslo(y). (V prípade, že platí Číslo(y) $>$ Číslo(x), môžeme pri analyzovaní situácie postupovať analogicky). Vypočítajme Rozd(x, y) = Číslo(x) $-$ Číslo(y)

[11] Napríklad pre $x = 10110$ je Číslo(x) $= 22$ a pre $p = 5$ je Číslo(x) $= 22 = 4 \cdot 5 + 2$, kde $h_x = 4$ a $s = 2$ je zvyšok.

$$\begin{array}{r} \text{Číslo}(x) \\ -\text{Číslo}(y) \\ \hline \text{Rozd}(x,y) \end{array} \qquad \begin{array}{r} h_x \cdot p + s \\ -h_y \cdot p - s \\ \hline h_x \cdot p - h_y \cdot p \end{array}$$

teda pre rozdiel $\text{Rozd}(x,y)$ platí:

$$\text{Rozd}(x,y) = \text{Číslo}(x) - \text{Číslo}(y) = h_x \cdot p - h_y \cdot p = (h_x - h_y) \cdot p.$$

Dôsledkom je, že p delí číslo $\text{Rozd}(x,y) = \text{Číslo}(x) - \text{Číslo}(y)$. Inými slovami:

Prvočíslo je zlé pre (x,y) práve vtedy, keď p delí číslo $\text{Rozd}(x,y) = \text{Číslo}(x) - \text{Číslo}(y)$.

Čo nám to pomôže? Našli sme rýchly spôsob ako určiť, či je prvočíslo zlé.

Príklad 6.2 Nech má počítač R_I reťazec bitov $x = 1001001$ a R_{II} má $y = 0101011$, oba dĺžky $n = 7$. Úlohou je nájsť množinu zlých prvočísiel.

Najprv určíme množinu prvočísiel:

$$\begin{aligned} \text{PRIM}(n^2) &= \text{PRIM}(49) \\ &= \{2,3,5,7,11,13,17,19,23,29,31,37,41,43,47\}, \end{aligned}$$

z ktorej sa v protokole SVEDOK náhodne vyberá prvočíslo. Podľa nášho zistenia sú pre $(1001001, 0101011)$ zlé práve tie prvočísla, ktoré delia rozdiel

$$\begin{aligned} \text{Rozd}(1001001, 0101011) &= \text{Číslo}(1001001) - \text{Číslo}(0101011) \\ &= 73 - 43 = 30. \end{aligned}$$

Okamžite vidíme, že 2, 3 a 5 sú jediné prvočísla z PRIM(49), ktoré delia 30, preto sú 2, 3 a 5 jediné zlé prvočísla pre $(1001001, 0101011)$. Takže dostávame, že

$$\text{Chyba}_{SVEDOK}(1001001, 0101011) = \frac{3}{\text{Prim}(49)} = \frac{3}{15} = \frac{1}{5}.$$

□

Úloha 6.5 Nájdite všetky zlé prvočísla pre nasledujúce dvojice reťazcov bitov.

(i) $(01010, 11101)$

(ii) $(110110, 010101)$

(iii) $(11010010, 01101001)$.

Ostáva nám odpovedať ešte na otázku:

> „Ako nám pomôže poznanie, že zlé prvočísla delia $\text{Rozd}(x, y)$ určiť ich počet?“

Pretože obe čísla $\text{Číslo}(x)$ aj $\text{Číslo}(y)$ sú menšie ako 2^n platí aj, že

$$\text{Rozd}(x, y) = \text{Číslo}(x) - \text{Číslo}(y) < 2^n.$$

Chceme ukázať, že číslo menšie ako 2^n môže mať najviac $n - 1$ prvočiniteľov (prvočísiel, ktoré ho delia). Aby sme to ukázali, využijeme inú známu vetu z hodín školskej matematiky - základnú vetu aritmetiky. Táto hovorí, že

> *každé prirodzené číslo sa dá jednoznačne vyjadriť ako súčin prvočísiel.*

Platí napríklad:

$$5\,940 = 2 \cdot 2 \cdot 3 \cdot 3 \cdot 3 \cdot 5 \cdot 11 = 2^2 \cdot 3^3 \cdot 5 \cdot 11 \ ,$$

a teda 5 940 má 4 prvočinitele $2, 3, 5$ a 11.

Nech sú $p_1, p_2, p_3, \ldots, p_k$ všetky prvočinitele nášho čísla $\text{Rozd}(x, y)$ a nech $p_1 < p_2 < p_3 < \ldots < p_k$. Platí, že $p_1 \geq 2$ (najmenšie prvočíslo). Vo všeobecnosti vidíme, že pre všetky i platí $p_i > i$. Takže dostávame:

$$\text{Rozd}(x, y) = p_1^{i_1} \cdot p_2^{i_2} \cdot p_3^{i_3} \cdot \ldots \cdot p_k^{i_k}$$

pre $i_j \geq 1$ pre všetky j od 1 po k a môžeme písať:

$$\begin{aligned} \text{Rozd}(x, y) &\geq p_1 \cdot p_2 \cdot p_3 \cdot \ldots \cdot p_k \\ &> 1 \cdot 2 \cdot 3 \cdot \ldots \cdot k \\ &= k! \end{aligned}$$

Keďže $\text{Rozd}(x, y) < 2^n$ a zároveň $\text{Rozd}(x, y) > k!$ (ako sme práve ukázali), dostaneme:

$$2^n > k! \ .$$

Pretože pre $n \geq 4$ platí, že $n! > 2^n$, musí byť k menšie ako n, čím sme dosiahli vytýčený cieľ, že

$$k \leq n - 1.$$

To znamená, že počet prvočiniteľov $\text{Rozd}(x, y)$ je najviac $n - 1$, a tak je počet zlých prvočísiel pre (x, y) najviac $n - 1$.

Konečne môžeme zhora ohraničiť pravdepodobnosť chyby protokolu SVEDOK pre $(x, y), x \neq y$. Pre všetky dvojice reťazcov bitov (x, y) dĺžky $n \geq 9$ platí

$$\begin{aligned} \text{Chyba}_{SVEDOK}(x, y) &= \frac{\text{počet zlých prvočísiel pre } (x, y)}{\text{Prvoč}(n^2)} \\ &\leq \frac{n-1}{n^2/\ln n^2} \\ &\leq \frac{2\ln n}{n}. \end{aligned}$$

Takže pravdepodobnosť chyby protokolu SVEDOK pre rôzne obsahy databáz x a y je najviac $2\ln n/n$, čo v našom prípade, keď $n = 10^{16}$ je najviac

$$\frac{0,73683}{10^{14}}.$$

V príklade 6.2 sme videli, že pravdepodobnosť chyby bola relatívne vysoká (1/5). Je to preto, lebo *pravdepodobnosť chyby sa zmenšuje, čím je vstup väčší*. Preto sa v prípade malých vstupných údajov s veľkosťou niekoľko tisíc bitov odporúča porovnávanie vykonať deterministicky, pretože náklady sú v takomto prípade tak, či tak zanedbateľné. Randomizovaný protokol sa oplatí použiť až v prípade dlhších vstupov.

Úloha 6.6 Aká je pravdepodobnosť chyby pre vstupné údaje dĺžky $n = 10^3$ a $n = 10^4$? Od akej dĺžky by ste dôverovali randomizovanému protokolu SVEDOK?

Úloha 6.7 Určte presne pravdepodobnosť chyby protokolu SVEDOK pre nasledujúce dvojice (x, y).

(i) $(00011011, 10101101)$
(ii) $(011000111, 000111111)$
(iii) $(0011101101, 0011110101)$.

Práve sme zažili zázrak. Randomizovaný postup si vie efektívne poradiť s úlohami, ktoré sa nedajú v praxi vyriešiť deterministicky kvôli nevyhnutným vysokým nákladom. Poznáme viaceré problémy, ktoré sa prakticky nedajú vyriešiť s deterministickými algoritmami, pretože aj najefektívnejšie z nich majú exponenciálnu zložitosť. Mnohé z nich riešime

v praxi denne a často to vieme úspešne vykonať len vďaka využitiu náhody. Celá e-komercia, a teda aj online banking a online nakupovanie by nebola možná bez softvéru, v ktorom sú implementované algoritmy využívajúce náhodu. Pre tých, čo hľadajú absolútnu istotu, je to vlastne zlá správa. Ale pre tých, ktorí sú si vedomí rizika, vedia s ním počítať a udržať ho malé, sú programy využívajúce náhodu geniálnym riešením.

Teraz je načase zbaviť zázrak kúzla. Radi by sme sa dopátrali, prečo využitie náhody vyčaruje takýto ohromný efekt a toto porozumenie by sme mali chápať ako normálne správanie sa vecí okolo nás.

Začnime naše úvahy spochybnením toho, čo sme doteraz predstavili. Videli sme, že protokol SVEDOK využívajúci náhodu pracuje skutočne efektívne a veľmi spoľahlivo. Sú naozaj pravdivé naše nedokázané tvrdenia, že najlepší deterministický protokol pre našu komunikačnú úlohu vyžaduje pre niektoré vstupy výmenu n bitov? A že pre väčšinu vstupov (x, y) je potrebných skoro n komunikačných bitov? Prečo by práve na tomto mieste nemohol čitateľ zapochybovať? Pretože tvrdenia ozaj vyzerajú podozrivo.

Poďme hlbšie do podstaty veci a pozrime sa ešte raz na obrázok 6.2. Dobré prvočísla pre (x, y) nazývame **svedkovia rozličnosti x od y.** Hovoríme, že prvočíslo p **dosvedčuje** skutočnosť $x \neq y$, keď platí

$$\text{Číslo}(x) \bmod p \neq \text{ Číslo}(y) \bmod p.$$

Prvočíslo q **nie je svedok** rozličnosti x a y, keď

$$\text{Číslo}(x) \bmod p = \text{ Číslo}(y) \bmod p.$$

Teda dobré prvočísla pre (x, y) sú svedkovia pre (x, y) a zlé prvočísla pre (x, y) **nie sú svedkovia pre (x, y)**. Po tomto premenovaní vidíme, že práca protokolu SVEDOK je založená na hľadaní svedkov toho, že „$x \neq y$“. Keď náhodne vyberieme svedka toho, že „$x \neq y$“, pokračujeme deterministicky ďalej a spoľahlivo dostaneme správny výsledok. Keď si vyberieme nie svedka, nevyhnutne prídeme k zlému výsledku. Celý protokol pracuje správne s vysokou pravdepodobnosťou preto, lebo v množine PRIM(n^2) sú skoro samí svedkovia. Veľkosť skupiny nie svedkov je oproti Prvoč(n^2) taká malá, že pravdepodobnosť náhodného výberu nie svedka je tiež veľmi malá. A teraz príde dôvod na pochybnosti.

> *Keď množina kandidátov na svedkov obsahuje len skoro samých svedkov, prečo si z nich nevieme deterministicky jedného vybrať a po krátkej komunikácii úlohu správne vyriešiť? Deterministicky si vybrať svedka znamená odstrániť úplne využitie*

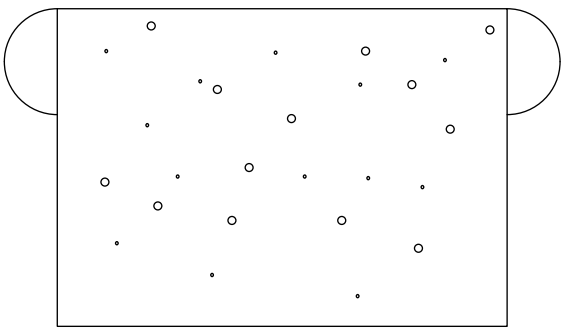
náhody a takto by sme mohli potenciálne vytvoriť na riešenie úlohy efektívny deterministický protokol.

Ako rozpoznáme, že tento na prvý pohľad prijateľný nápad sa nedá uskutočniť? Prirodzene, môžeme poskytnúť matematický dôkaz neexistencie efektívneho deterministického protokolu na riešenie tejto úlohy. To vedie k dvom problémom. Tento dôkaz nie je ťažký pre skúseného informatika a matematika, ale vyžaduje vedomosti, ktoré nemôžeme očakávať u všeobecného publika. A po druhé ani pochopenie dôkazu nepomôže porozumieť, *prečo* predkladaná idea derandomizácie nefunguje. Z jeho argumentácie vyplýva, že sa to nedá, ale skutočné dôvody zostanú nevyjasnené a tajomné. Preto sa pokúsime len o intuitívne priblíženia sa k podstate problému a pritom si prehĺbiť naše chápanie využitia náhody.

Neexistuje efektívny[12] deterministický algoritmus, ktorý by určil svedka, lebo z pohľadu oboch počítačov R_I aj R_{II}

sú svedkovia „náhodne“ rozptýlení medzi všetkými kandidátmi.

Čo to znamená presnejšie? Keď poznáte x, ale o y viete len veľmi málo, nemusíte vystačiť pri hľadaní svedka pre (x, y) ani s viacerými pokusmi. Je to preto, lebo pre rôzne vstupy dĺžky n sú svedkovia v množine kandidátov $\mathrm{PRIM}(n^2)$ rozdelení úplne inak. Neexistuje pravidlo, podľa ktorého by sme s obmedzenými vedomosťami o vstupe vedeli vypočítať svedka. Preto vyzerá z pohľadu počítačov R_I a R_{II} obsah nádoby s kandidátmi ako pravý chaos (chaotická zmes svedkov a nesvedkov). Práve v tom väzí jadro úspechu využitia náhody.



obr. 6.3: Chaotická zmes svedkov a nesvedkov.

Vedeli by sme efektívne vyriešiť viaceré problémy, keby sme mali svedkov správnej odpovede. V praxi ale nijakých svedkov nedostaneme. Keď ale

[12] vzhľadom na komunikačnú zložitosť

vieme vytvoriť množinu kandidátov na svedkov a vieme, že v nej je veľa svedkov, hoci aj chaoticky sa vyskytujúcich, stojí za to vyloviť niekoho náhodne. Keď je rozdelenie svedkov medzi kandidátmi ozaj náhodné a javí sa preto ako chaotické, nemá nijaký deterministický postup šancu nájsť svedka efektívne.

Pri hľadaní efektívnych algoritmov využívajúcich náhodu využívame často nasledujúci postup. Hľadáme takých svedkov, ktorí majú tieto tri vlastnosti:

(i) Keď máme pre vstup svedka, vieme efektívne deterministicky vypočítať správny výsledok.

(ii) Keď máme pre vstup kandidáta na svedka, vieme efektívne preveriť, či je svedok alebo nie.

(iii) Svedkovia sa hojne vyskytujú v množine kandidátov.

Na základe vlastnosti (iii) nazývame túto metódu návrhu algoritmov využívajúcich náhodu **metóda hojnosti svedkov**. Veľký počet svedkov je dobrá vec. Ako podľa potreby zvýšiť ich výskyt, je téma nasledujúceho odseku.

6.4 Čo môžeme urobiť, keď zákazník požaduje vysokú mieru istoty?

Celá história ľudstva je spätá s hľadaním istoty. Čo sme? Čo tu hľadáme? Čo máme urobiť, aby sme si zabezpečili „peknú“ budúcnosť alebo aspoň nejakú budúcnosť? Život a veda nás poučili, že hľadanie absolútnej istoty je nezmyselné, ba môže sa stať osudným. Usilovať sa o neexistujúcu absolútnu istotu znamená aj mnohého sa nezmyselne zriecť a často skončiť v slepej uličke. Typické následky sú rezignácia, frustrácia a depresie. Keď ale akceptujeme neistotu a spolu s ňou aj náhodu ako neoddeliteľnú súčasť života a naučíme sa s ňou žiť, objavíme namiesto frustrácie možnosti, ktoré sa nám otvoria po vzdaní sa neexistujúcich istôt. Slepé uličky viac nebudú slepými uličkami a budúcnosť bude mať nepreberné množstvo podôb. Presne tak to bolo aj v našom príklade z predchádzajúcej časti a ešte častejšie je aj v algoritmike. Situácia vyzerajúca na pohľad beznádejne, pretože každý protokol riešiaci našu úlohu má príliš vysoké komunikačné nároky. Výmena nedosiahnuteľnej ideálnej absolútnej istoty za praktickú istotu poskytuje dobré možnosti na riešenie. V iných podobných situáciách sa nám nemusí vždy bezpodmienečne podariť nájsť

početných svedkov ako v našom komunikačnom protokole. Niekedy tvoria naozajstní svedkovia len zlomok počtu všetkých kandidátov na svedkov. V takých prípadoch sa môže zvýšiť aj pravdepodobnosť chyby, a to až natoľko, že je to v praxi neakceptovateľné. Cieľom nasledujúceho odseku je ukázať, ako úspešne zvládnuť takéto situácie.

Začnime s príkladom nášho komunikačného protokolu. Pre $n = 10^{16}$ bola pravdepodobnosť chyby najviac $0.74 \cdot 10^{-14}$. Predstavme si, že máme zákazníka, pre ktorého je správnosť výsledku protokolu enormne dôležitá a želá si ešte vyššiu mieru istoty. Napríklad povie:

> *„Chcem aby bola pravdepodobnosť chyby taká malá, že keď protokol použijem toľkokrát, koľko je vek vesmíru v sekundách vynásobený počtom protónov vo vesmíre, tak s pravdepodobnosťou* $1 : 1\,000\,000$ *nedostanem nesprávny výsledok ani v jednom z tohto ohromného počtu pokusov.“*

Vieme splniť takéto prepiate požiadavky bez extrémneho zväčšenia zložitosti? Odpoveď je „ÁNO“ a prv, ako si ukážeme ako na to, urobíme si malú jednoduchú úvahu z teórie pravdepodobnosti.

Predstavme si, že máme korektnú hraciu kocku. Hádzanie kockou chápeme ako náhodný experiment, v ktorom je možných presne 6 výsledkov 1, 2, 3, 4, 5 a 6 a je rovnako pravdepodobné, že nastane ľubovoľný z nich, teda pravdepodobnosť každého je presne 1/6.

Predpokladajme ďalej, že jediný nepriaznivý výsledok je pre nás, keď padne „1“. Všetky ostatné výsledky sú priaznivé. Preto je pravdepodobnosť nepriaznivého výsledku 1/6. Zmeňme teraz pravidlá. Hodíme kocku päťkrát za sebou a jediný nepriaznivý výsledok je, keď padnú samé „1“. Inak povedané, sme spokojní, keď z piatich vrhov aspoň raz padne na kocke niečo iné ako „1“. Aká veľká je pravdepodobnosť, že nastane nepriaznivý výsledok? Máme teda určiť, aká veľká je pravdepodobnosť, že päťkrát za sebou padne „1“. Pretože tieto pokusy považujeme za navzájom nezávislé[13], počítame to takto. Jednu „1“ hodíme s pravdepodobnosťou 1/6. Dve „1“ za sebou hodíme s pravdepodobnosťou

$$\frac{1}{6} \cdot \frac{1}{6} = \frac{1}{36}.$$

Pravdepodobnosť toho, že päťkrát za sebou hodíme „1“ je

$$\frac{1}{6} \cdot \frac{1}{6} \cdot \frac{1}{6} \cdot \frac{1}{6} \cdot \frac{1}{6} = \frac{1}{6^5} = \frac{1}{7\,776}.$$

[13] Podrobné vysvetlenie, čo znamená nezávislé, môžeme nájsť v [Hro04b].

Čo majú spoločné tieto úvahy s naším protokolom? Veľa. Na protokol využívajúci náhodu môžeme nazerať ako na náhodný experiment, v ktorom nepriaznivý výsledok zodpovedá náhodnému výberu nesvedka pre daný vstup. Výskumom protokolu sme zistili, že pravdepodobnosť nepriaznivého výsledku, a teda aj chyby je najviac

$$\frac{2 \ln n}{n}.$$

Postupujme analogicky ako pri kockách! Keď vyberieme náhodne po sebe 10 prvočísiel, je jediný nepriaznivý výsledok, keď sa pri všetkých desiatich prvočíslach nenájde svedok rôznosti medzi x a y. Desaťkrát za sebou vybrať prvočíslo vedie k nasledujúcemu protokolu využívajúcom náhodu.

SVEDOK (10)
Východisková situácia: Počítač R_I má n bitov $x = x_1 \dots x_n$ a počítač R_{II} má n bitov $y = y_1 \dots y_n$.

1. fáza: R_I si zvolí náhodne 10 prvočísiel

$$p_1, p_2 \dots p_{10} \text{ z PRIM}(n^2).$$

2. fáza: Pre všetky i od 1 po 10 vypočíta R_I

$$s_i = \text{Číslo}(x) \bmod p_i$$

a pošle binárnu reprezentáciu čísiel

$$p_1, p_2 \dots p_{10}, s_1, s_2 \dots s_{10}$$

počítaču R_{II}.

3. fáza: Po obdržaní 20 čísiel $p_1, p_2 \dots p_{10}, s_1, s_2 \dots s_{10}$ vypočíta R_{II}

$$q_i = \text{Číslo}(y) \bmod p_i$$

pre všetky $i = 1, 2 \dots 10$.
V prípade, že aspoň pre jedno i z $1, 2, \dots 10$ dostaneme, že $q_i \neq s_i$, vie R_{II} s istotou, že $x \neq y$ a dá výsledok „sú rôzne“.
Keď $q_i = s_j$ pre všetkých desať j z $1, 2 \dots 10$, potom buď $x = y$ alebo $x \neq y$ a nijaké z desiatich zvolených prvočísiel nie je svedkom toho, že $x \neq y$. V tomto prípade R_{II} odpovie „sú rovnaké“.

Čo sme si mohli všimnúť? Keď platí $x = y$, odpovie SVEDOK(10) tak isto ako SVEDOK zaručene správny výsledok „sú rovnaké“. V prípade,

že platí $x \neq y$, môže odpovedať SVEDOK(10) nesprávne iba v prípade, keď nijaké z desiatich prvočísiel, ktoré si zvolil, nebolo svedkom pre x a y. Ak je z desiatich pokusov čo len jeden svedkom pre x a y, napríklad p_4, potom $s_4 \neq q_4$ svedčí o tom, že x a y nie sú rovnaké a je zaručená správna odpoveď „sú rôzne“. Keď je pravdepodobnosť toho, že v jednom pokuse nevyberieme svedka najviac $2 \ln n/n$, je pravdepodobnosť toho, že svedka pre x a y nevyberieme v 10 pokusoch, v najhoršom

$$\left(\frac{2 \ln n}{n}\right)^{10} = \frac{2^{10} \cdot (\ln n)^{10}}{n^{10}}.$$

Pre $n = 10^{16}$ je to najviac

$$\frac{0,4\,717}{10^{141}}.$$

Počet sekúnd od veľkého tresku vynásobený počtom protónov vo vesmíre je menší ako

$$10^{99}.$$

Vynecháme matematické zdôvodnenie toho, že Šanca, že pri 10^{99} použitiach SVEDOK(10) dostaneme nesprávnu odpoveď je menšia ako 1 ku 10^{30}.

Pravdepodobnosť chybnej odpovede sme tým stlačili hlboko pod hranicu všetkých praktických požiadaviek na obmedzenie výskytu chyby. A za tento zázrak platíme veľmi málo. Keď si náhodne zvolíme 10 prvočísiel namiesto jedného, komunikačné náklady sa zdesaťnásobia, čo znamená, že pre SVEDOK(10) budú

$$40 \cdot \lceil \log_2 n \rceil .$$

Tieto náklady sú zanedbateľné. Vidno to, keď pre $n = 10^{16}$ zistíme, že SVEDOK(10) potrebuje komunikovať len

$$2\,560 \text{ bitov.}$$

Pretože SVEDOK(10) je desať opakovaní protokolu SVEDOK, v informatike hovoríme, že

> *so zvýšením počtu opakovaní (pokusov nájsť svedka) rastie zložitosť lineárne a pravdepodobnosť chyby sa zmenšuje exponenciálne.*

Je to vlastne asi najpriaznivejšia situácia, aká vôbec môže nastať.

Náš algoritmus SVEDOK(10) využívajúci náhodu je prakticky spoľahlivejší ako hocičo, čo môžeme asociovať s pojmom bežná spoľahlivosť. Je

to preto, lebo už aj SVEDOK bol vlastne veľmi spoľahlivý. Metódou opakovaných pokusov dokážeme pravdepodobnosť chybného výsledku podstatne znížiť aj v prípade, keď je neveľká pravdepodobnosť, že svedok bude vybraný spomedzi kandidátov v jednom pokuse. V nasledujúcich cvičeniach prenecháme čitateľovi, aby použil predstavené myšlienky, a prehĺbil si tak pochopenie sily randomizácie.

Úloha 6.8 Nech je A úloha, pre ktorú všetky známe deterministické algoritmy na jej riešenie majú zložitosť aspoň 2^n. Obsahom úlohy A je rozhodnúť, či vstup x má alebo nemá určitú vlastnosť. Pre x veľkosti n máme spôsob svedčenia s nasledujúcimi vlastnosťami:

(i) Keď je z svedkom pre x, potom sa vieme v čase potrebnom na vykonanie $10 \cdot n^2$ operácií presvedčiť, že x má želanú vlastnosť. V prípade, že z nie je svedkom, algoritmus nie je schopný určiť, či x má alebo nemá želanú vlastnosť. Na určenie, že z nie je svedkom pre x potrebuje algoritmus tiež čas $10 \cdot n^2$.
(ii) Počet svedkov v množine kandidátov je aspoň polovica (t.j. pravdepodobnosť, že vyberieme svedka je aspoň $1/2$).

Zadanie úlohy:

1. Navrhnite algoritmus využívajúci náhodu, ktorý sa 20 krát pokúsi vybrať svedka z množiny kandidátov. Aká je zložitosť a aká je pravdepodobnosť chyby algoritmu?
2. Zákazník si želá algoritmus, ktorý vyrieši úlohu A s pravdepodobnosťou chyby najviac 1 k 10^9. Koľko pokusov na výber svedka nevyhnutne potrebujeme? Aká je zložitosť takéhoto algoritmu?

Úloha 6.9 Vyriešte úlohu 6.8 v prípade, keď je splnený predpoklad (i) a namiesto (ii) platí jedna z nasledujúcich podmienok:

a) Podiel svedkov v množine kandidátov je presne $1/6$.
b) Podiel svedkov v množine kandidátov je iba jedna n-tina, pričom n je veľkosť vstupných údajov.

6.5 Čo sme objavili?

Keď trváme na stopercentnej teoretickej záruke, že vypočítané riešenie jednotlivých prípadov daných úloh je vždy správne, môžeme sa ocitnúť v bezvýchodiskovej situácii. Všetky algoritmy, spĺňajúce požiadavku absolútnej spoľahlivosti, vyžadujú také výpočtové náklady, ktoré ich robia prakticky neuskutočniteľnými. Poznáme príklady problémov, pri ktorých najlepšie známe algoritmy vyžadujú viac času ako je vek vesmíru a viac

energie ako je v celom vesmíre. Riešenie takýchto problémov v prípade požadovanej absolútnej spoľahlivosti je mimo fyzikálne realizovateľného. A v tejto zdanlivo bezvýchodiskovej situácii zistíme, že môžeme urobiť zázrak. Zriekneme sa tak, či tak nedosiahnuteľnej absolútnej spoľahlivosti výpočtu a budeme požadovať iba, aby sme vypočítali s vysokou pravdepodobnosťou správny výsledok pre každý prípad problému. Smieme pritom požadovať tak malú pravdepodobnosť chyby, že môžeme hovoriť o zámene hypoteticky absolútnej spoľahlivosti za praktickú spoľahlivosť. Týmto iba zdanlivým rezignovaním na maximálne požiadavky, pokiaľ ide o korektnosť riešenia, vieme ušetriť obrovské množstvo práce počítača. Pri niektorých problémoch dosiahneme vďaka šikovnému využitiu náhody v algoritme na ich riešenie skok od fyzikálne nerealizovateľného množstva výpočtov k výpočtom, ktoré sa dajú za pár sekúnd vykonať na bežnom počítači.

Naučili sme sa dve dôležité paradigmy návrhu systémov využívajúcich náhodu. Metóda hojných (početných) svedkov odhalila najhlbšie korene sily randomizácie. Svedok pre konkrétny prípad problému predstavuje dodatočnú informáciu, pomocou ktorej tento prípad vieme efektívne vyriešiť, čo by sme bez svedka neboli schopní. Metóda má dobré vyhliadky na úspech, keď vieme nájsť taký spôsob svedčenia, pri ktorom sa svedkovia hojne vyskytujú medzi kandidátmi. Vtedy vieme náhodným (podľa potreby aj opakovaným) výberom s vysokou pravdepodobnosťou získať z množiny kandidátov svedka, napriek tomu, že nevidíme možnosť efektívne určiť svedka deterministicky. Základnú otázku, či sú ťažké problémy, ktoré vieme efektívne riešiť randomizovane, ale nedajú sa riešiť deterministicky, môžeme vyjadriť rečou metódy hojných svedkov takto:

> *V prípade, že pre ťažké výpočtové problémy máme len množiny kandidátov na svedkov, v ktorých sú náhodne rozptýlení (rozmiestnení) často sa vyskytujúci svedkovia náhodne, sú tieto problémy efektívne riešiteľné randomizovane, ale nie sú efektívne riešiteľné deterministickými algoritmami.*

Druhý dôležitý princíp návrhu randomizovaných algoritmov, ktorý sme sa naučili, je paradigma *Zvýšenia pravdepodobnosti úspechu (rýchle zmenšenie pravdepodobnosti chyby) opakovaným spustením navrhnutého systému využívajúceho náhodu.* Desaťnásobné opakovanie protokolu SVEDOK vyústilo do protokolu SVEDOK(10), ktorého pravdepodobnosť chyby sa približovala k nule exponenciálne rýchlo vzhľadom na počet opakovaní. Pritom zložitosť rástla vzhľadom na počet opakovaní len lineárne. Situácia nie je vždy taká priaznivá, ale väčšinou sme schopní bez veľkého zníženia

efektívnosti dosiahnuť (zákazníkom) požadovanú mieru spoľahlivosti.

Kniha „Randomisierte Algorithmen“ alebo „Design and Analysis of Randomized Algorithms“ [Hro04b, Hro05] podrobne predstavuje metodiku tvorby systémov využívajúcich náhodu pre začiatočníkov. Na jednej strane sa snaží malými krokmi vytvoriť intuitívne pochopenie predmetu, na druhej strane používa dôsledne rigorózny jazyk matematiky, aby mohla poskytnúť jednoznačné zdôvodnenie všetkých tvrdení a výsledkov analýzy algoritmov. Najpodrobnejšie a najhlbšie predstavenie témy randomizácie nájdeme v monografii „Randomized Algorithms“ [MR95] od Motwaniho a Raghavana, ktorá je veľmi náročná a je určená predovšetkým výskumníkom a študentom tejto oblasti. Lahôdkou pre labužníkov je kniha „Geschichte der Algorithmenentwicklung für den Primzahltest“ od Martina Dietzfelbingera [Die04], ktorá je ovplyvnená konceptmi randomizácie, najmä použitia metódy hojných svedkov. Ďalšie efektívne komunikačné protokoly využívajúce náhodu obsahujú knihy [KN97, Hro97]. Nájdete v nich prezentáciu matematicky dokázateľných výhod randomizácie oproti deterministickému prístupu. Žiaľ, tieto knihy sú matematicky náročné, a preto ťažko prístupné pre širšie publikum.

Návody riešenia vybraných úloh

Úloha 6.2 Nijaké prvočíslo okrem 7 z PRIM(25), nevedie k chybnej odpovedi „sú rovnaké“ pri použití SVEDOK pre $x = 01111$ a $y = 10110$.

Úloha 6.5 (i) Nech $x = 01010$ a $y = 11101$. Preto platí $n = 5$ a do úvahy berieme prvočísla z PRIM(5^2). Čísla zodpovedajúce x a y sú:

$$\begin{aligned} \text{Číslo}(01010) &= 2^3 + 2^1 = 8 + 2 = 10 \\ \text{Číslo}(11101) &= 2^4 + 2^3 + 2^2 + 2^0 = 16 + 8 + 4 + 1 = 29. \end{aligned}$$

Aby sme určili zlé prvočísla z PRIM(25), nemusíme skúšať všetky prvočísla. Vieme, že každé zle prvočíslo musí deliť rozdiel:

$$\text{Číslo}(11101) - \text{Číslo}(01010) = 29 - 10 = 19.$$

Číslo 19 je prvočíslo, takže jediné prvočíslo z PRIM(25) deliace 19, je samotná 19. Preto je 19 jediné zlé prvočíslo pre 01010 a 11101.

Úloha 6.7 (i) Aby sme spočítali pravdepodobnosť chyby protokolu SVEDOK pre $x = 00011011$ a $y = 10101101$, musíme určiť veľkosť PRIM(8^2) a počet zlých prvočísiel pre x a y v PRIM(8^2).

$$\text{PRIM}(64) = \{2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61\}$$

a platí, že |PRIM(64)| = 18. Ďalej platí, že

$$\begin{aligned} \text{Číslo}(x) &= 2^4 + 2^3 + 2^1 + 2^0 = 16 + 8 + 2 + 1 = 27 \\ \text{Číslo}(y) &= 2^7 + 2^5 + 2^3 + 2^2 + 2^0 = 128 + 32 + 8 + 4 + 1 = 173. \end{aligned}$$

Takže Číslo(y) − Číslo(x) = 173 − 27 = 146. Číslo 146 sa dá rozložiť na prvočinitele:

$$146 = 2 \cdot 73 \ .$$

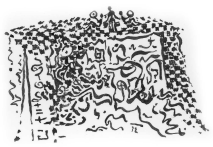
Teda jediné prvočísla, ktoré delia Číslo(y) − Číslo(x) = 146, sú čísla 2 a 73. Pretože 73 sa nenachádza v PRIM(64), je v PRIM(64) presne jedno zlé prvočíslo pre x a y. Pravdepodobnosť chyby je preto presne 1/18.

Úloha 6.8

1. Algoritmus sa skončí, keď nájde svedka pre x. V najhoršom prípade urobí 20 pokusov a každý pokus stojí najviac $10 \cdot n^2$ operácií potrebných na preverenie, či vybraný objekt je svedok. Takže zložitosť algoritmu využívajúceho náhodu je v najhoršom prípade $20 \cdot 10 \cdot n^2$. Keď vstup x nemá hľadanú vlastnosť, dá algoritmus s istotou správnu odpoveď „NIE“. Keď x má vlastnosť, môže algoritmus odpovedať nesprávne „NIE“ iba vtedy, keď 20-krát po sebe nevyberie svedka. Pravdepodobnosť, že pri pokuse nevyberieme svedka, je najviac 1/2 (podľa (ii)). Pravdepodobnosť, že v 20 po sebe idúcich pokusoch nevytiahneme svedka, je teda najviac

   $$\left(\frac{1}{2}\right)^{20} = \frac{1}{2^{20}} = \frac{1}{1\,048\,576} < 0,000\,001 \ .$$

   Takže pravdepodobnosť chyby nie je menšia ako 1 k miliónu.
2. Koľko pokusov potrebujeme, aby sme stlačili pravdepodobnosť pod $1/10^9$? Vieme, že $2^{10} = 1\,024 > 1\,000$. Takže $2^{30} = (2^{10})^3 > 1\,000^3 = 10^9$. Teda stačí 30 pokusov, aby bola pravdepodobnosť chybnej odpovede menej ako 1 k miliarde.



Z geniálnej myšlienky sa dajú odstrániť všetky slová.
Stanisław Jerzy Lec

Kapitola 7

Kryptografia alebo ako spraviť zo slabiny prednosť

7.1 Magická veda súčasnosti

Najfascinujúcejšou vedou dvadsiateho storočia bola pravdepodobne fyzika. Priniesla hlboké a fundamentálne poznatky, ktoré podstatným spôsobom zmenili náš pohľad na svet. Mnohé aplikácie a interpretácie fyzikálnych objavov, akými sú teória relativity alebo kvantová mechanika, pôsobia ako zázraky. Málokto veril, že niečo také je vôbec možné. Podľa môjho názoru najmagickejšou vedeckou disciplínou súčasnosti je kryptografia. Verili by ste, že:

- každý sa môže presvedčiť o tom, že vlastníte tajomstvo bez toho, aby ste z neho čo len zlomok prezradili?
- šifrovaná správa sa dá poslať viacerým príjemcom tak, že títo ju môžu dešifrovať a prečítať iba keď všetci bez výnimky spolupracujú (spoluvytvoria tajný kľúč)?
- sa dá elektronicky podpísať dokument tak, aby sa každý mohol presvedčiť o jeho pravosti, ale nikto nevedel elektronický podpis sfalšovať?
- vo verejnom rozhovore sa dá s inou osobou dohodnúť na tajnom kľúči bez toho, aby sa prípadný načúvajúci o ňom čokoľvek dozvedel?

Všetko uvedené a mnoho ďalšieho je možné. Jednoducho čistá „mágia“!

Komunikácia je heslom dňa Nezáleží na tom, Či ide o mobilné telefóny, sms-ky, internet, elektronickú poštu, alebo klasický fax, písanie listu alebo telefonovanie, všade sa komunikuje alebo sa komunikačnými sieťami prenášajú údaje. Pri takomto množstve odosielaných správ ustavične rastú nároky na ochranu súkromia zúčastnených, keďže odoslané správy by nemal byť schopný prečítať nikto okrem prijímateľa, ktorému boli určené. Nároky na bezpečnosť stúpajú predovšetkým pri elektronickom nákupe a predaji[1], peňažných transakciách[2] alebo elektronických voľbách.

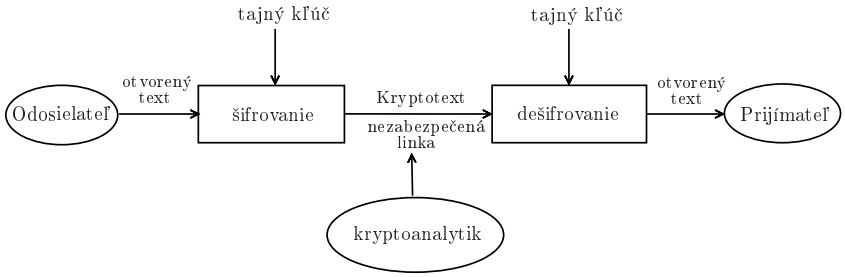
Kryptológia pôvodne znamenala náuku o tajných písmach a je skoro taká stará ako samotné písmo. V kryptológii rozlišujeme medzi **kryptografiou** a **kryptoanalýzou**. Kryptografia sa zapodieva návrhom kryptosystémov (tajných kódov) a kryptoanalýza študuje spôsoby, ako kryptosystémy prelomiť (tajne čítať správy). Kedysi dávno, keď ľudia objavili a zdokonalili písmo, vedelo písať a čítať iba pár zasvätených. V tomto úzkom kruhu ľudí ho mohli používať namiesto šifrovaného písma. Ako rástol počet ľudí, ktorí vedeli čítať a písať, rástla aj potreba po ozajstnom šifrovanom spôsobe písania. V tejto kapitole sa najprv krátko obzrieme do histórie, keď sa kryptológia ešte považovala za umenie. Vyvíjali sa šifry, používali sa dovtedy, kým niekto neprišiel na spôsob, ako šifrované správy prečítať, čím príslušná šifra prestala byť bezpečná. V krátkosti sa pozrieme, ako šifrovali správy Caesar a Richelieu. Pritom sa naučíme koncept tvorby kryptosystému (šifrovaného písma) a čo rozumieme pod spoľahlivosťou alebo bezpečnosťou kryptosystému.

Naším ďalším cieľom bude ukázať, ako sa vďaka informatike (najmä algoritmike a teórii zložitosti) umenie vytvárať šifry zmenilo na vednú disciplínu, ktorá sa nazýva kryptografia. Kryptografia vďačí informatike za základný pojem: bezpečnosť kryptosystému (axiomatický základný kameň), ktorý umožňuje vedecký výskum v tejto oblasti. Vďaka nemu sa vyvinula moderná kryptografia, založená na magickom koncepte takzvaných Public-Key-kryptosystémov[3]. Vysvetlíme hlavnú myšlienku tohto konceptu a ukážeme, ako môžeme využiť vlastné slabé stránky v náš prospech. Na základe teórie zložitosti vieme, že existujú problémy, ktoré sa nedajú efektívne algoritmicky vyriešiť (študovali sme ich v kapitole 5). Tieto využijeme pri vytváraní kryptosystémov, ktoré sa prakticky nedajú ohroziť. A nakoniec si pozrieme ešte pár divov, ako sa dá tento koncept magicky využiť.

[1] e-commerce
[2] online-banking
[3] kryptosystémov s verejnými kľúčmi

7.2 Koncept kryptosystému alebo prehistória kryptológie

Najprv uvedieme odborné pojmy. Ako sme už spomínali, **kryptológiu** chápeme ako náuku o tajných písmach. V rámci kryptológie rozlišujeme medzi kryptografiou, vedeckou disciplínou zaoberajúcou sa vytváraním kryptosystémov a kryptoanalýzou, umením tieto prelomiť. V ďalšom sa sústredíme na kryptografiu. Náš scenár je na obrázku obr. 7.1.



obr. 7.1

Osoba, nazývaná **odosielateľ**, posiela tajnú správu druhej osobe, ktorú voláme **prijímateľ**. Tajná správa má podobu textu, budeme ju označovať ako **otvorený text**. Otvorený text je napísaný v prirodzenom jazyku, a je teda pre každého dobre čitateľný. Správu doručuje nie celkom stopercentne spoľahlivý posol, alebo sa posiela nechráneným verejným komunikačným kanálom. Aby sme zabránili niekomu nepovolanému, kto by sa akokoľvek k správe dostal, prečítať jej tajný obsah, pošleme správu v **šifrovanej** forme[4]. Spôsob **šifrovania** a **dešifrovania** je spoločným tajomstvom odosielateľa a prijímateľa a šifrovanie sa vykonáva pomocou takzvaného **kľúča**. Zašifrovaný text voláme **kryptotext**. Po prijatí prijímateľ kryptotext **dešifruje**. Dešifrovaním sa z kryptotextu vytvorí opäť pôvodný otvorený text, ktorý prijímateľ môže bez problémov čítať. Opis procesu šifrovania a dešifrovania poskytuje úplnú informáciu o kryptosystéme, umožňujúcu nám ho použiť.

Pre lepšie znázornenie sa pozrime na kryptosystém CAESAR, ktorý Caesar skutočne používal. Otvorené texty boli texty napísané v latinskej abecede, z ktorej boli vynechané prázdne znaky, čiarky, otázniky, bodky atď. Takže

VENIVIDIVICI

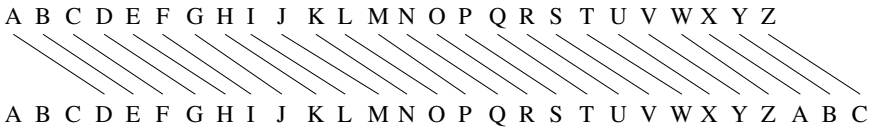
[4] tajným písmom

môžeme považovať za otvorený text. Caesar šifroval každé písmeno jednoznačne za nejaké iné písmeno. Kľúč bol daný jedným číslom od 1 po 26.

0, 1, 2, 3, 4, 5, ..., 23, 24, 25

Použiť kľúč i znamená, nahradiť každé písmeno písmenom, ktoré je od neho v abecede o i miest neskôr. Obrázok 7.2 znázorňuje kódovanie kľúčom $i = 3$. To znamená, že písmeno A nahradíme s D, lebo A je na prvej pozícii v abecede a posunom o tri pozície dostaneme štvrtú pozíciu, na ktorej je písmeno D. Analogicky sa písmeno B z druhej pozície zakóduje písmenom E z piatej pozície, atď. Týmto spôsobom sa dostaneme až k písmenu W na 23. pozícii, ktoré sa zakóduje písmenom Z z 26. pozície. Čo sa ale stane so zvyšnými tromi písmenami X, Y a Z, ktoré sú na pozíciách 24, 25 a 26? Na zakódovanie nám ostali ešte písmená A, B a C. Napíšeme si teda A B C na pozície 27, 28 a 29 za spodnú abecedu na obr. 7.2 a pokračujeme v kódovaní rovnakým spôsobom ako predtým. Teda X sa zakóduje ako A, Y ako B a Z ako C. Iný spôsob ako si tento proces znázorniť, je takýto: Predstavme si, že máme kruhový pás, ktorý je nastoknutý na pevnom hriadeli, okolo ktorého sa môže otáčať. Abecedu napíšeme na otočný pás a na pevný hriadeľ si napíšeme zodpovedajúce pozície písmen v abecede. Keď pás otočíme o tri pozície, dostaneme hľadané kódovanie písmen. Pre kľúč $i = 3$ sa správa VENIVIDIVICI zakóduje ako:

YHQLYLGLYLFL



obr. 7.2

obr. 7.3

Úloha 7.1 a) Zakódujte Cézarovým kryptosystémom text VENIVIDIVICI kľúčom $i = 5$ a potom $i = 17$.
b) Zakódujte otvorený text „Roma aeterna docet“ kľúčom $i = 7$ a otvorený text „Homines sumus, non dei!“ kľúčom $i = 13$.

Ako dekóduje príjemca kryptotext? Jednoducho, každé písmeno kryptotextu nahradí písmenom, ktoré sa nachádza v abecede o tri pozície skôr. Zodpovedá to otočeniu pásu na obr. 7.3 o tri pozície doľava. Pritom sa napríklad písmeno S kryptotextu nahradí písmenom P, písmeno U sa nahradí písmenom R a L sa nahradí I. Teda keď príjemca dešifruje kryptotext

SULPXPQHQRFHDV

kľúčom $i = 3$, dostane otvorený text

PRIMUMNENOCEAS

Úloha 7.2 Dešifrujte kryptotext WYPTBTBAWYVMPJLHZ kľúčom $i = 7$.

Úprimne povedané, Caesar nechcel zbytočne riskovať, že niekto takýto relatívne jednoduchý kryptosystém pochopí, a preto v kryptotexte (po posunutí písmen) nahradil písmená v latinke písmenami gréckej abecedy. Takže prijímateľ musel najskôr nahradiť grécke písmená, aby dostal príslušný kryptotext v latinke, a až potom posunúť pomocou kľúča abecedu naspäť.

Jules Verne vo svojich románoch používa zovšeobecnený Cézarov kryptosystém. Kľúč mohlo byť ľubovoľné celé číslo. Pre celé číslo s m ciframi

$$A = a_1 a_2 \dots a_m$$

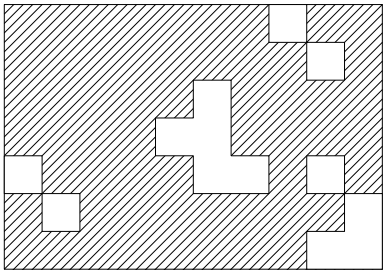
rozdelil text na bloky dĺžky m a j-te písmeno v každom bloku nahradil písmenom, ktoré je v abecede o a_j pozícií za ním. Napríklad pre $a = 316$ a otvorený text

C	R	Y	P	T	O	G	R	A	P	H	Y
3	1	6	3	1	6	3	1	6	3	1	6,

dostaneme cryptotext

FSESUUJSGSIE

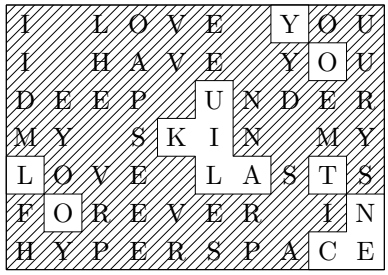
Na rozdiel od vecného postupu Cézarovho šifrovania, Richelieov spôsob šifrovania by sa dal označiť ako umenie. Nasledujúci príklad použitia RICHELIEU kryptosystému pochádza od Salomaa [Sal96]. Kryptotext je



(a)

I		L	O	V	E		Y	O	U
I		H	A	V	E		Y	O	U
D	E	E	P		U	N	D	E	R
M	Y		S	K	I	N		M	Y
L	O	V	E		L	A	S	T	S
F	O	R	E	V	E	R		I	N
H	Y	P	E	R	S	P	A	C	E

(b)



(c)

obr. 7.4

obyčajný popísaný list papiera. Každé písmeno má svoje presné miesto, napríklad H leží v druhom riadku a treťom stĺpci na obr. 7.4 (b). Kľúč je matica na obr. 7.4 (a), v ktorej sú určité (sivé) políčka zakryté a určité otvorené. Cez otvorené políčka vidíme otvorený text. (obr. 7.4 (c))

YOUKILLATONCE.

7.3 Kedy je kryptosystém bezpečný?

Ľudia sa ustavične snažia dosiahnuť bezpečnosť. Dôvody ich úzkosti sú často pocity neistoty a nebezpečna. No veda a najmä fyzika nás učia, že absolútna istota neexistuje. Bezhlavo sa o ňu usilovať môže spôsobiť až zdravotné problémy. Lepšie je naučiť sa žiť s neistotou. Napriek tomu má ale zmysel usilovať sa o dosiahnuteľné bezpečnostné záruky. Kedy budeme považovať kryptosystém za „bezpečný“? Vždy vtedy, keď náš protivník, alebo niekto nepovolaný nebude vedieť na základe kryptotextu zrekonštruovať zodpovedajúci otvorený text. Táto požiadavka má dve možné interpretácie. Malo by byť ťažké, ak nie nemožné, aby kryptoanalytik dešifroval kryptotext, keď nevie nič o kryptosystéme, alebo aj vtedy, ak je umenie ako text zašifrovať známe a neznámy je iba šifrovací kľúč? Prvú možnosť, keď uchovávame v tajnosti spôsob šifrovania, nazývame „Security by Obscurity“ a nepovažuje sa za dostatočnú na definovanie bezpečnosti kryptosystému. Dôvodom je naša skúsenosť, že je to vždy len otázka času, kým niekto príde na spôsob, ako sa v novom systéme šifruje.

Preto už v 19. storočí sformuloval Auguste Kerckhoffs nasledujúcu bezpečnostnú požiadavku, ktorú poznáme ako **Kerckhoffsov princíp**:

> *Kryptosystém je bezpečný, keď z prijatého kryptotextu nevieme bez znalosti kľúča odvodiť pôvodný otvorený text, hoci spôsob šifrovania je verejne známy.*

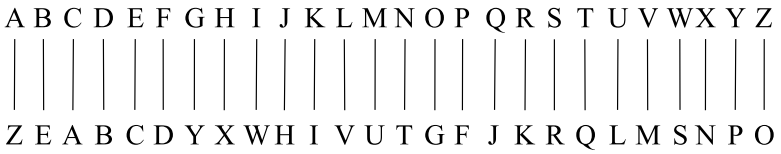
Kryptosystém CAESAR nie je navidomoči v uvedenom zmysle bezpečný. Keď je známy jeho princíp, tak na to, aby sme rozšifrovali ľubovoľný kryptotext, stačí vyskúšať všetkých 25 možných kľúčov.

Z tohto príkladu môžeme hneď usúdiť, že bezpečný kryptosystém musí mať na výber z veľkého počtu kľúčov. Stačí to ale naozaj? Vylepšime systém CAESAR tým, že dovolíme vytvárať ľubovoľné dvojice písmen. Kľúčom bude teraz takzvaná permutácia z 26 písmen, ktorú si môžeme predstaviť ako 26-ticu čísiel od 1 po 26, pričom každé číslo od 1 po 26 sa v nej vyskytuje iba raz. Napríklad nasledujúci kľúč:

> (26, 5, 1, 2, 3, 4, 25, 24, 23, 8, 9, 22, 21, 20, 7, 6, 10, 11, 18, 17, 12, 13, 19, 14, 16, 15)

zodpovedá šifrovaniu, pri ktorom sa písmeno A nahradí 26. písmenom - Z, písmeno B sa nahradí piatym písmenom - E, písmeno C písmenom A, atď. obr. 7.5 znázorňuje zodpovedajúce dvojice písmen.

Keď použijeme tento kľúč, dostaneme kryptotext:



obr. 7.5

EJZFWLZQZMEBWNZMCXBEJCQUMFKXBEJWKSZAYSCLC
QM

z nasledujúceho otvoreného textu:

CHODSVOJOUCESTOUANECHAJLUDINECHSIROZPRAVAJU

Úloha 7.3 Dešifrujte nasledujúce dva kryptotexty, ktoré vznikli použitím kľúča z obr. 7.5.

a) BZTQCZVWYXWCKW
b) YCBLVBWRQBCKRAXVLCRRCVDLCKDKCLBC

Koľko kľúčov má tento kryptosystém? Na zakódovanie písmena A máme na výber z 26 písmen. Po tom, čo si zvolíme zašifrovanie písmena A, máme ešte 25 možností ako zakódovať B. Pre písmeno C máme 24 možností, atď. Takže počet rôznych kľúčov je:

$$26! = 26 \cdot 25 \cdot 24 \cdot 23 \cdot \ldots \cdot 3 \cdot 2 \cdot 1,$$

čo je (v reči kombinatoriky) počet všetkých permutácií 26 prvkov. Je to ozaj obrovské číslo, približne $4,03 \cdot 10^{26}$. Takýto počet kľúčov nik nemôže preskúšať ani s pomocou najrýchlejších počítačov.

Je teda už kryptosystém bezpečný? Odpoveď je, žiaľ, jednoznačná: „nie“. Na to, aby sme objavili ten správny kľúč, nemusíme nevyhnutne vyskúšať všetky možné. Postačí nám šikovná myšlienka, ktorá neprišla na um tvorcovi kryptosystému a systém poľahky prelomíme. V prípade šifrovania permutáciou písmen stačí vedieť, v ktorom prirodzenom jazyku sa komunikuje. Pre každý prirodzený jazyk je známe, ako často sa vyskytujú jednotlivé písmená v slovách a tieto početnosti sú navzájom dosť rôzne. Okrem toho sa niektoré kombinácie písmen vyskytujú častejšie, napríklad v nemčine je to SS alebo SCH. Toto kryptoanalytikovi stačí na to, aby kryptotext rozšifroval, a teda aj určil tajný kľúč.

Uvedený príklad ilustruje, ako ťažko je formálne uchopiť pojem „nebyť schopný, alebo len veľmi ťažko dešifrovať“. V Kerkhoffsovej definícii bezpečnosti je navyše jeden stupeň voľnosti. Nejde iba o samotný kryptosystém, ale aj o protivníka. Čo je nemožné pre niektorého protivníka, môže byť pre iného hračkou. Čo môžeme o protivníkovi predpokladať, alebo ako dostať všetkých protivníkov pod spoločný klobúk?

Dlhotrvajúcu nejasnosť pojmu bezpečnosť v kryptológii ukončila informatika. Zjednotila všetky rovnako jednoduché, ako aj geniálne, stratégie kryptoanalytikov pod pojem algoritmus.

> *Kryptosystém považujeme za bezpečný, keď neexistuje efektívny (randomizovaný polynomiálny) algoritmus, ktorý na základe poznania kryptotextu a spôsobu šifrovania bez tajného kľúča vypočíta pôvodný otvorený text.*

Je zrejmé, že takáto definícia bezpečnosti kryptosystému nemohla vzniknúť pred zavedením pojmu algoritmus.[5] Tým vieme vyjadriť prvé dve dôležité požiadavky na „dobrý“ kryptosystém v terminológii teórie zložitosti.

(i) Šifrovanie a dešifrovanie vieme pri známom kľúči realizovať efektívnym algoritmom.

(ii) Dešifrovanie bez kľúča predstavuje ťažký (prakticky neriešiteľný) problém.

7.4 Symetrické kryptosystémy

Kryptosystémy, o ktorých bola doposiaľ reč, nazývame **symetrické**. Slovo symetrické znamená, že odosielateľ i prijímateľ správy sú rovnocenní v tom zmysle, že rovnaký tajný kľúč sa použije na zašifrovanie aj dešifrovanie správy. V princípe by si mohol odosielateľ s prijímateľom hocikedy vzájomne vymeniť úlohy bez toho, aby zmenili kľúč. Ich komunikáciu môžeme teda chápať ako rozhovor alebo vzájomnú výmenu informácií.

Teraz si ukážeme bezpečný symetrický komunikačný systém. Pre účely tohto kryptosystému budeme chápať otvorený text ako postupnosť núl a jednotiek. Nijako nás to neobmedzí, pretože každé písmeno a ľubovoľný

[5] Zavedeniu pojmu algoritmus vďačíme za viacero podobne podstatných pokrokov vo vede.

symbol, ktorý vieme napísať na klávesnici, má ASCII kód, ktorý je binárnou reprezentáciou príslušného symbolu v počítači. Takto sa dá každý text premeniť na postupnosť núl a jednotiek.

Ako kľúč použijeme tiež postupnosť núl a jednotiek, napríklad zoberieme niekoľko sto náhodne vygenerovaných bitov, aby sme prípadnému tretiemu sťažili, alebo mu až úplne znemožnili kľúč nejakým spôsobom vypočítať.

Skôr ako definujeme spôsob šifrovania a dešifrovania, musíme sa naučiť jednu operáciu s bitmi (binárnymi číslami). Táto operácia sa nazýva „exkluzívne alebo“ alebo „XOR“, označuje sa $\oplus$ a definujeme ju nasledujúco:

$$0 \oplus 0 = 0 \qquad 1 \oplus 1 = 0 \qquad 0 \oplus 1 = 1 \qquad 1 \oplus 0 = 1$$

Na základe označenia zo 6. kapitoly platí $a \oplus b = a + b \bmod 2$. Dva rovnaké sčítance dávajú nulu a dva rôzne dávajú jednotku.

Predpokladajme, že kľúč je 01001011. Kľúč má dĺžku 8 a môžeme ho použiť na zašifrovanie 8 bitov otvoreného textu. Pritom použijeme i-ty bit kľúča a operáciou $\oplus$ zašifrujeme i-ty bit otvoreného textu. Názorne to vidieť, keď si kľúč napíšeme pod otvorený text:

	otvorený text	00001111
$\oplus$	kľúč	01001011
=	kryptotext	01000100

i-ty bit kryptotextu je XOR-súčet i-teho bitu otvoreného textu a i-teho bitu kľúča. Teraz použijeme rovnaký postup na rozšifrovanie:

	kryptotext	01000100
$\oplus$	kľúč	01001011
=	otvorený text	00001111

Vidíme, že to funguje a dostaneme pôvodný otvorený text. Je to založené na skutočnosti, že

$$a \oplus a = 0 \quad \text{a teda platí} \quad b \oplus a \oplus a = b$$

Dvojnásobné použitie kľúča nás privedie k pôvodnému textu. Inak povedané: Druhé použitie kľúča zruší jeho prvé použitie. Okrem toho platí aj:

$$a \oplus b = b \oplus a.$$

Preto hovoríme, že tento spôsob šifrovania je komutatívny.

Úloha 7.4 Zašifrujte a dešifrujte otvorený text 0001110011 s kľúčom 0101101001.

Úloha 7.5 Vytvorte podobný kryptosystém, ktorý bude založený na nasledujúcej maskovacej binárnej operácii:

$$0 \perp 0 = 1 \qquad 1 \perp 0 = 0 \qquad 1 \perp 1 = 1 \qquad 0 \perp 1 = 0$$

Druhý bit b v $a \perp b$ nazývame maskovací. Keď $b = 1$, tak sa prvý bit skopíruje. A v prípade keď $b = 0$, sa prvý bit a preklopí na $\bar{a}$ (teda 1 na $\bar{1} = 0$ a 0 na $\bar{0} = 1$).

a) Použite operáciu $\perp$ na zašifrovanie a dešifrovanie otvoreného textu 00110011 kľúčom 00101101.
b) Zdôvodnite, prečo aj kryptosystém založený na operácii $\perp$ funguje.
c) Čo majú operácie $\perp$ a $\oplus$ spoločné? Je medzi nimi úzky vzťah?

Ak je na otvorený text použitý ako „maska“ kľúč, ktorý náhodne generujú odosielateľ aj príjímateľ[6], dá sa matematicky zdôvodniť, prečo sa kryptotext každému inému zdá ako náhodná postupnosť bitov. V takom prípade nepomôžu kryptoanalytikom ani hypoteticky použiteľné exponenciálne algoritmy a najrýchlejšie počítače. Takýto kryptosystém môžeme považovať pri jednorazovom použití ako bezpečný. Keď máme dlhý otvorený text, šifrujeme ho zvyčajne podobným spôsobom kľúčom s pevnou dĺžkou n, pričom otvorený text rozdelíme na úseky dĺžky n a každý z nich zašifrujeme kľúčom osobitne. Napríklad keď je kľúč 0101, rozdelíme otvorený text

1111001100001101

na štyri časti 1111, 0011, 0000 a 1101, a každú časť zašifrujeme samostatne a odošleme postupnosť vzniknutých kryptotextov

1010011001011000.

To je typický spôsob použitia kryptosystémov, vytvorených pre otvorené texty pevnej dĺžky.

V prípade XOR-kryptosystémov sa takéto rozšírenie neodporúča, pretože pri opakovanom použití rovnakého kľúča je možné ho určiť. Platí:

$$\text{otvorený text} \oplus \text{kryptotext} = \text{kľúč} \tag{7.1}$$

Preveríme, či to platí na našom prvom príklade s kľúčom 01001011.

[6] Teda ho nie je potrebné prenášať.

	otvorený text	00001111
$\oplus$	kryptotext	01000100
=	kľúč	01001011

Prečo je to tak?

$$\text{otvorený text} \oplus \text{kľúč} = \text{kryptotext} \tag{7.2}$$

zodpovedá šifrovaniu. Teraz pripočítajme zľava k obom stranám rovnice (7.2) otvorený text a dostaneme:

$$\text{otvorený text} \oplus \text{otvorený text} \oplus \text{kľúč} = \text{otvorený text} \oplus \text{kryptotext} \tag{7.3}$$

Keďže $a \oplus a = 0$ pre každý bit a

$$\text{otvorený text} \oplus \text{otvorený text} = 00\ldots00.$$

A keďže $0 \oplus b = b$ pre každý bit b, ľavá strana rovnice (7.3) sa rovná kľúču, čím dostávame rovnicu (7.1).

Rovnica (7.1) je pre bezpečnosť kryptosystému veľmi nebezpečná. Keby sa pri viacnásobnom použití rovnakého kľúča dostal nepriateľovi nejako do rúk čo len jediný pár (otvorený text, kryptotext), okamžite by využitím (7.1) vedel vypočítať kľúč a dešifrovať všetky nasledujúce kryptotexty. Z tohto, ale nielen z tohto dôvodu je XOR-kryptosystém bezpečný iba pri jednorazovom použití kľúča.

Úloha 7.6 (tvrdý oriešok) Pokúste sa vytvoriť bezpečný kryptosystém pre nasledujúcu úlohu. Osoba A zašifruje otvorený text kľúčom, ktorý pozná len ona (tajná správa). Správa je určená pre dvoch ľudí B a C. Osoba A chce tajný kľúč rozdeliť medzi B a C tak, aby ani jeden z nich nebol schopný kryptotext sám dešifrovať a ani zistiť čo len jediný bit otvoreného textu. Keď ale B a C budú spolupracovať, kryptotext bez problémov rozšifrujú.

Sú aj iné symetrické kryptosystémy, ktoré sú bezpečné a pri ktorých dokonca bez váhania môžeme kľúč opakovane použiť. Najznámejší a v súčasnosti najrozšírenejší symetrický kryptosystém je DES (Data Encryption Standard), ktorý bol vyvinutý spoločne IBM a NSA (National Security Agency). Používa sa v ňom okrem iných aj operácia XOR, ale je príliš zložitý na to, aby sme ho tu opísali.

Napriek doteraz spoľahlivému použitiu niekoľkých symetrických kryptosystémov nie sme ešte na konci nášho hľadania bezpečného systému. Problém je v tom, že symetrické kryptosystémy sa dajú spoľahlivo použiť iba

v prípade, keď sa odosielateľ a prijímateľ dopredu dohodnú na spoločnom kľúči. Ale ako to majú urobiť bez toho aby sa stretli? Ako sa majú na začiatku bez kryptosystému a prostredníctvom nie bezpečného komunikačného kanála dohovoriť na spoločnom kľúči? Riešenie tohto problému je predmetom nasledujúcej podkapitoly.

7.5 Zhodnutie sa na spoločnom kľúči cez nechránený verejný komunikačný kanál

Dve osoby, Alica a Bob, chcú vytvoriť spoločný symetrický kryptosystém. Spôsob šifrovania je obom známy, potrebujú sa iba dohodnúť na spoločnom kľúči. Pretože sa ale nemôžu stretnúť osobne, musia sa dohodnúť na tomto spoločnom tajomstve bez kryptosystému a len využitím verejného komunikačného kanála, ktorý môže každý odpočúvať. Je to vôbec možné?

Odpoveď je „áno“, a je prekvapujúce ako elegantne sa dá tento problém vyriešiť. Ukážeme si najprv názorne hlavnú myšlienku pomocou truhlice, a nie stopercentne spoľahlivého posla. Alica použije zámok, od ktorého má kľúč iba ona a analogicky Bob použije zámok, ktorý vie otvoriť len on.[7] Alica sa s Bobom dohodne, že budúci spoločný kľúč vloží do truhlice a pošle Bobovi. Postupujú nasledujúco:

1. Alica vloží tajný kľúč do truhlice a uzamkne ju svojim zámkom. Okrem nej truhlicu nemôže nik otvoriť, lebo iba ona má správny kľúč od svojho zámku. Truhlicu pošle Bobovi.

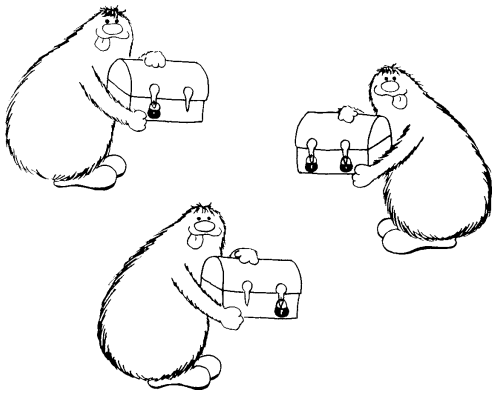
2. Posol prinesie truhlicu Bobovi, ktorý ju samozrejme tiež nevie otvoriť. Namiesto toho, aby sa ju pokúšal otvoriť, aj Bob zamkne truhlicu jeho zámkom. Truhlicu uzamkutú dvomi zámkami pošle naspäť Alici. Teraz nik nemôže truhlicu otvoriť.

3. Alica dostane truhlicu s dvomi zámkami. Odomkne ju a odstráni svoj zámok. Na truhlici zostal iba Bobov zámok, a tak pošle truhlicu znova Bobovi.

4. Bob dostane truhlicu, zamknutú iba jeho zámkom. Otvorí ho a z truhlice si prevezme spoločný tajný kľúč.

Obrázok 7.6 z [Sal96] zábavným spôsobom znázorňuje uvedenú procedúru.

[7] Nemajú ešte spoločný kľúč.



obr. 7.6

Úloha 7.7 (tvrdý oriešok) Alica chce rovnaký tajný kľúč bezpečne poslať trom ďalším osobám. Jedna možnosť je zopakovať trikrát horeopísanú procedúru. Pritom bude musieť posol bežať $3 \cdot 3 = 9$ krát medzi dvoma ľuďmi. Dá sa uskutočniť odovzdanie kľúča trom osobám tak, aby posol musel bežať menejkrát?

Dá sa tento proces realizovať aj elektronicky? Skúsme to najprv s operáciou XOR. Týmto spôsobom dostaneme veľmi jednoduchú implementáciu (realizáciu) truhlice s dvomi zámkami. Ako neskôr uvidíme, táto implementácia ešte nespĺňa všetky bezpečnostné kritériá. Na obr. 7.7 je znázornený priebeh komunikácie medzi odosielateľom a príjemcom.

Odosielateľ (Alica) chce príjemcovi oznámiť kľúč, ktorý je znázornený na obr. 7.7 ako otvorený text. Otvorený text je postupnosť núl a jednotiek. Súkromné kľúče odosielateľa a príjemcu sú tiež postupnosti núl a jednotiek a majú rovnakú dĺžku ako otvorený text. Postup má tri komunikačné kroky, preto ho v kryptografii nazývame **komunikačný protokol**. Priebeh protokolu je nasledujúci:

1. Odosielateľ vypočíta

   otvorený text $\oplus$ kľúč A = krypto 1

   a pošle krypto 1 príjemcovi.

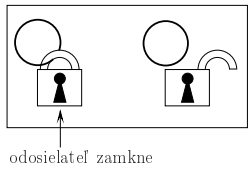
2. Príjemca vypočíta

   krypto 1 $\oplus$ kľúč B = krypto 2

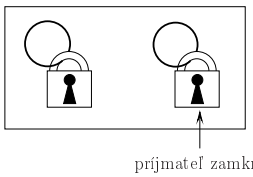
   a pošle krypto 2 naspäť odosielateľovi.

3. Odosielateľ vypočíta

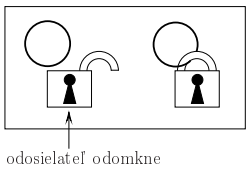
   krypto 2 $\oplus$ kľúč A = krypto 3



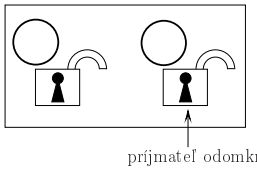
	otvorený text	101011
$\oplus$	kľúč odosielateľa	011011
	prvý kryptotext	110000



	prvý kryptotext	110000
$\oplus$	kľúč príjemcu	101010
	druhý kryptotext	011010



	druhý kryptotext	011010
$\oplus$	kľúč odosielateľa	011011
	tretí kryptotext	000001



	tretí kryptotext	000001
$\oplus$	kľúč príjemcu	101010
	otvorený text	101011

obr. 7.7

[všimnime si, že platí

krypto 3 = krypto 2 $\oplus$ kľúč A
= krypto 1 $\oplus$ kľúč B $\oplus$ kľúč A
{lebo krypto 2 = krypto 1 $\oplus$ kľúč B}
= otvorený text $\oplus$ kľúč A $\oplus$ kľúč B $\oplus$ kľúč A
{lebo krypto 1 = otvorený text $\oplus$ kľúč A}
= otvorený text $\oplus$ kľúč A $\oplus$ kľúč A $\oplus$ kľúč B
{vďaka komutatívnosti $\oplus$ môžeme
vymeniť poradie argumentov}
= otvorený text $\oplus$ kľúč B
{lebo $a \oplus a = 0$ a $b \oplus 0 = b$}.

]

4. príjemca vypočíta

$$\begin{aligned} \text{krypto } 3 \oplus \text{kľúč B} &= \text{otvorený text} \oplus \text{kľúč B} \oplus \text{kľúč B} \\ &\quad \{\text{pretože, ako sme pred chvíľou ukázali,} \\ &\quad \text{krypto } 3 = \text{otvorený text} \oplus \text{kľúč B}\} \\ &= \text{otvorený text} \end{aligned}$$

Úloha 7.8 Zahrajte sa tak, že vyskúšate priebeh komunikačného protokolu na bezpečnú výmenu kľúča pre nasledujúce údaje: otvorený text a zároveň aj budúci tajný kľúč je 01001101. Kľúč odosielateľa je 01010101 a kľúč príjemcu je 10101010.

Pri opise komunikačného protokolu na výmenu kľúča sme zdôvodnili, prečo takto príjemca na konci skutočne dostane odoslaný kľúč. Hlavná myšlienka spočíva v tom, že druhé aplikovanie kľúča automaticky zruší prvé aplikovanie aj keby boli medzi týmito aplikáciami iné akcie. Môžeme to zapísať takto:

$$\begin{aligned} \text{text} \oplus \text{kľúč} \oplus \text{akcie} \oplus \text{kľúč} &= \text{text} \oplus \text{akcie} \oplus \underbrace{\text{kľúč} \oplus \text{kľúč}}_{\text{zruší sa}} \\ &= \text{text} \oplus \text{akcie}. \end{aligned}$$

Úloha 7.9 Opísaný protokol funguje, pretože operácia $\oplus$ má pekné vlastnosti $a \oplus a = 0$, $a \oplus b = b \oplus a$ a $b \oplus 0 = b$. Preskúmajte, či sa dá tento komunikačný protokol implementovať pomocou operácie $\perp$.

Je tento komunikačný protokol bezpečný? Poskytuje rovnakú záruku bezpečnosti ako truhlica s dvoma zámkami? Bohužiaľ nie. V prípade, že kryptoanalytik nepozná postup a získa iba jednotlivé kryptotexty, ktoré sa v rámci protokolu posielajú, nevie ich odlíšiť od náhodných postupností bitov. V tomto zmysle je naša implementácia postupu bezpečná. Podľa Kerkhoffsovho princípu musíme ale počítať s tým, že protivník pozná komunikačný protokol a jediné, čo nevie, sú dva tajné náhodne vygenerované kľúče. Ako vidno z nasledujúceho výpočtu, keď ale získa všetky tri kryptotexty (obr. 7.7), vie určiť kľúče odosielateľa i príjemcu

$$\begin{aligned} \text{kľúč príjemcu} &= \text{prvý kryptotext} \oplus \text{druhý kryptotext} \\ \text{kľúč odosielateľa} &= \text{druhý kryptotext} \oplus \text{tretí kryptotext} \end{aligned}$$

Úloha 7.10 Využitím vlastností operácie $\oplus$ overte platnosť predchádzajúcich rovníc na výpočet kľúčov odosielateľa a príjemcu.

Tým je prezradené celé tajomstvo, lebo stačí jeden z nich, aby sme dešifrovali otvorený text:

$$\begin{array}{lcrcl} \text{otvorený text} & = & \text{kľúč odosielateľa} & \oplus & \text{prvý kryptotext} \\ \text{otvorený text} & = & \text{kľúč príjemcu} & \oplus & \text{tretí kryptotext} \end{array}$$

Ako vidieť, náš komunikačný protokol nie je veľmi bezpečný. Nedal by sa nejako náš „fyzický“ postup s truhlicou a dvoma zámkami preniesť do digitálneho sveta pri zachovaní rovnakej miery bezpečnosti? Túto otázku zodpovedali kladne v roku 1976 Whitfield Diffie a Martin Hellman [DH76]. Použili pri tom šikovným spôsobom počítanie modulo prvočíslo, s ktorým sme narábali už v Kapitole 6. Opíšeme tento postup podobným spôsobom ako na obr. 7.7 bez toho, aby sme zachádzali do podrobného matematického zdôvodnenia jeho správnosti. Pokiaľ nemáte k matematike vzťah, môžete nasledujúci opis protokolu preskočiť.

Komunikačný protokol Diffie-Hellman

Východisková situácia: odosielateľ a príjemca sa otvorene dohodnú na dvoch veľkých prirodzených číslach c a p, pričom p je prvočíslo a platí, že $c < p$.

Odosielateľ náhodne vygeneruje číslo a_{ODO} a toto číslo bude jeho tajný súkromný kľúč.

Príjemca vygeneruje náhodne číslo a_{PRI} a to bude tajný kľúč príjemcu, ktorý nik okrem neho nepozná.

Spoločná úloha príjemcu a odosielateľa je po vzájomnej komunikácii vypočítať nový kľúč, ktorý bude ich spoločným tajomstvom a ktorý obaja použijú pri symetrickom šifrovaní.

Postup

1. Odosielateľ vypočíta
   Kryptotext 1 $= c^{a_{ODO}} \bmod p$
   a pošle číslo kryptotext 1 príjemcovi.
2. Príjemca vypočíta
   Kryptotext 2 $= c^{a_{PRI}} \bmod p$
   a pošle kryptotext 2 odosielateľovi.
3. Odosielateľ vypočíta
   $S_A =$ (kryptotext 2)$^{a_{ODO}} \bmod p$
   a S_A považuje za nový spoločný kľúč.

4. Príjemca vypočíta
   $S_B = (\text{kryptotext } 1)^{a_{PRI}} \bmod p$
   a S_B považuje za nový spoločný kľúč.

Jadrom úspechu je, že platí $S_A = S_B$. Neuvedieme presné matematické zdôvodnenie. Vidíme ale, že S_A nie je nič iné ako c zašifrované najprv s a_{PRI} a potom s a_{ODO}. Kľúč S_B je tiež zašifrované c najprv s a_{ODO} a potom s a_{PRI}. Teda sú oba S_A aj S_B zašifrované s a_{ODO} a s a_{PRI}, len v opačnom poradí. To je také, ako keď v jednom prípade truhlicu zamkneme najprv ľavým a potom pravým zámkom a v druhom prípade najprv pravým a potom ľavým zámkom. Je očividné, že v prípade truhlice je výsledok rovnaký. Matematická funkcia $c^a \bmod p$ tu bola zvolená preto, aby ani poradie použitia súkromných kľúčov a_{ODO} a a_{PRI} nehralo úlohu.

Ak sú odosielateľov i príjemcov kľúč uchované v tajnosti, tak je spôsob šifrovania Diffie-Hellmana podľa našich doterajších kritérií bezpečný[8]. Práve pri pojme bezpečnosť musíme byť ale opatrnejší. Jasne sme sformulovali, čo považujeme za bezpečný kryptosystém v prípade posielania správy spôsobom CAESAR alebo DES. Pri komunikačných protokoloch, ktoré využívajú viacnásobnú výmenu informácií, sa musíme ešte raz zamyslieť nad pojmom bezpečnosť.

Doteraz sme predstavovali **pasívneho** protivníka (kryptoanalytika). Mohol načúvať, dozvedieť sa kryptotext a potom sa pokúšať rozlúsknuť ho. Voči takémuto pasívnemu protivníkovi je naša výmena kľúčov bezpečná. Ale všetko je inak, keď máme do činenia s **aktívnym protivníkom**, ktorý sa nazýva aktívny preto, lebo vstupuje do komunikácie. Predstavte si nasledujúci scenár. Protivník prehovorí posla, aby truhlicu doniesol jemu namiesto príjemcovi. Alebo preruší vedenie tak, že kryptotext 1 sa dostane iba k nemu, a nie k príjemcovi. Potom protivník zamkne truhlicu svojím zámkom a pošle ju naspäť odosielateľovi. Ten nevie, že namiesto pravého príjemcu komunikuje s protivníkom, otvorí svoj zámok a pošle príjemcovi truhlicu, zamknutú zámkom protivníka. Nespoľahlivý posol ju ale znovu zanesie protivníkovi, ktorý si ju odomkne a získa tajomstvo. Odosielateľ pritom nemá najmenšie podozrenie, že tajomstvo je prezradené.

Vidíme, že náš protokol ešte nie je vôbec perfektný, ale potrebuje ďalšie vylepšenia. Či sa vieme vysporiadať aj s aktívnym protivníkom, bude témou v nasledujúcom odseku.

[8] Nie je známy efektívny algoritmus, ktorým by sa dal vypočítať $S_A = S_B$ z daných dvoch kryptotextov a z c a p bez toho, aby sme poznali kľúč odosielateľa alebo príjemcu.

7.6 Kryptosystémy s verejnými kľúčmi

Najprv vymenujeme slabé miesta doteraz predstavených symetrických kryptosystémov.

(i) Symetrické kryptosystémy vyžadujú počiatočnú bezpečnú výmenu kľúčov. Voči aktívnemu protivníkovi ju nevieme spoľahlivo zabezpečiť.

(ii) V praxi sa do komunikácie často zapája veľa účastníkov. Informácie od mnohých sa zbierajú v centrále. Ak majú všetci rovnaký kľúč, stačí jediný zradca a celý systém je odhalený. Keď má každý z nich vlastný kľúč, treba viesť administratívu o mnohých kľúčoch a navyše pri posielaní správy odosielateľ musí vydať napospas svoju identitu.

(iii) Mnohé komunikačné úlohy sa nedajú realizovať symetrickými kryptosystémami. Pri elektronických voľbách sa chceme preukázať ako právoplatný volič, ale pri samotnom hlasovaní sa nechceme vzdať anonymity. Potrebujeme protokoly, ktorými môžeme preukázať kontrole našu právomoc (tým, že vlastníme nejaké tajomstvo, napríklad doklad totožnosti, alebo heslo) bez toho, aby sme odhalili čo i len jeden bit tohto tajomstva.

Tieto a iné príčiny si vyžiadali ďalší intenzívny výskum v kryptografii. Pri hľadaní riešenia nám opäť pomohli algoritmika a teória zložitosti, pričom sme našu slabinu, totiž neschopnosť riešiť ťažké problémy, premenili na na našu kryptografickú silnú stránku. Hlavná myšlienka je založená na existencii takzvaných jednosmerných funkcií, spomedzi ktorých máme niekoľko kandidátov, ktorí prichádzajú do úvahy. Ako **jednosmernú funkciu** označujeme funkciu f s nasledujúcimi vlastnosťami:

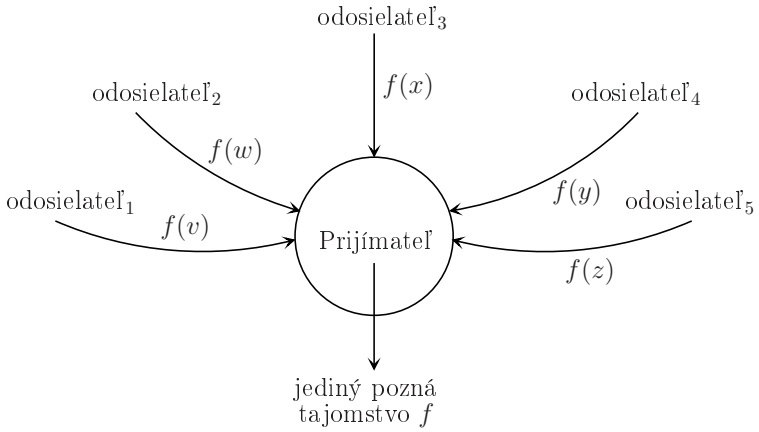
(i) Funkcia f sa dá efektívne vypočítať, teda sa dá použiť na efektívny spôsob šifrovania.

(ii) Inverzná funkcia f^{-1}, ktorá z hodnoty $f(x)$ vypočíta spätne argument x ($f^{-1}(f(x)) = x$) sa nedá efektívne vypočítať; teda neexistuje efektívny (randomizovaný) algoritmus, ktorým z daného kryptotextu = f(otvorený text), vypočítame argument otvorený text. Takže je na základe našej definície $f(x)$ kryptotext bezpečný.

(iii) Pre príjemcu ale musí existovať možnosť, ako z $f(x)$ efektívne vypočíta x. Teda o funkcii f musí existovať nejaké tajomstvo (niečo podobné ako bol svedok pri randomizovaných algoritmoch), na základe ktorého sa z $f(x)$ dá x rýchlo určiť.

Čo nám pomôže hypotetická jednosmerná funkcia f? Príjemca pozná o f určité tajomstvo, o ktoré sa nemusí počas komunikácie s nikým deliť (obr. 7.8). Funkcia f, a tým aj spôsob šifrovania môžu byť verejne známe a k dispozícii každému potenciálnemu odosielateľovi. Preto nazývame kryptosystémy, ktoré využívajú jednosmerné funkcie **kryptosystémy s verejným kľúčom** (Public-Key-Cryptosystem). Je zrejmé, že pri tomto spôsobe odpadne problém s výmenou tajných kľúčov, lebo nepotrebujeme mať spoločné tajomstvo. Týmto sme odstránili slabé miesta (i) a (ii) symetrických kryptosystémov. Navyše kryptosystémy s verejným kľúčom spĺňajú aj želanie (iii), ale detailné vysvetlenie tejto skutočnosti presahuje rámec tejto knihy.

Zatiaľ všetko sedí. Keď sa nám podarí nájsť jednosmerné funkcie, dosiahneme vytúžený cieľ. Existujú ale vôbec? Nevyzerajú tri požiadavky na takéto funkcie prepiate a neprirodzené? V nasledujúcom príklade sa vás pokúsim presvedčiť, že náš nápad nie je vôbec scestný.



obr. 7.8

Uvažujme o nasledujúcom spôsobe šifrovania. Každé písmeno jednoznačne nahradíme postupnosťou 14 desiatkových cifier. Pre každé písmeno vyberieme náhodne z nejakého telefónneho zoznamu meno začínajúce týmto písmenom a do kryptotextu zapíšeme zodpovedajúce telefónne číslo. V prípade, že číslo má menej ako 14 cifier, doplníme ho na začiatku príslušným počtom núl. Podľa tohto návodu dostaneme pre slovo KRYPTOGRAFIA napríklad:

	meno	telefónne číslo
K	Knuth	00128143752946
R	Rivest	00173411020745
Y	Yao	00127345912233
P	Papadimitriou	00372453008122
T	Thomas	00492417738429
O	Ogden	00012739226541
G	Good	00015402316555
R	Rabin	00048327450028
A	Adleman	00173555248001
F	Floyd	00013782442358
I	Ibarra	00124327010098
A	Aho	00183274553211

Predpokladajme, že okrem prijímateľa správy majú všetci len klasické telefónne zoznamy, ktoré sú abecedne usporiadané podľa priezvisk. Pomocou takýchto zoznamov by bolo veľmi náročné zistiť k jednotlivým telefónnym číslam v kryptotexte príslušné priezviská. Prijímateľ správy má k dispozícii celosvetový telefónny zoznam, utriedený podľa telefónnych čísiel, vďaka ktorému vie kryptotext efektívne rozšifrovať. Iné riešenie, ktoré príde na um často ako prvé, je zavolať na každé číslo. Okrem toho, že tento spôsob vyžaduje finančné náklady, nemáme nijakú istotu, že zistíme skutočné meno majiteľa telefónnej prípojky.

Tento príklad ilustruje len myšlienku. Uvedený spôsob nechceme považovať za ozajstný kryptosystém, pretože pre počítač nie je usporiadanie telefónneho zoznamu podľa čísiel nijako ťažká úloha. Ako jednosmernú funkciu budeme potrebovať inú operáciu.

Akých kandidátov na jednosmerné funkcie máme v praxi? Predstavíme najprv tri funkcie, ktoré spĺňajú podmienky (i) a (ii).

1. **Násobenie**
   Hocikto vie poľahky vynásobiť dve prvočísla p a q, $f(p,q) = p \cdot q$. Vypočítať ale z $f(p,q)$ naspäť hodnoty p a q je ťažká úloha, na riešenie ktorej existujú len exponenciálne algoritmy.

2. **Modulárna druhá mocnina**
   Je ľahké vypočítať funkciu $f_n(x) = x^2 \bmod n$. Umocníme x na druhú, dostaneme x^2, to vydelíme n a určíme zvyšok po delení, $x^2 \bmod n$. V prípade keď n nie je prvočíslo, je algoritmicky ťažké určiť hodnotu x na základe známych hodnôt $f_n(x)$ a n.

3. **Modulárne umocňovanie**
   Pre známe čísla e a n a otvorený text c (považujeme ho za číslo) ľahko vypočítame číslo $a = c^e \bmod n$. Keď n nie je prvočíslo, je algoritmicky ťažké spätne určiť otvorený text c zo známych hodnôt a, e a n.

Zdôvodnenie, že kryptosystémy s verejným kľúčom pracujú korektne, vyžaduje určité vedomosti z teórie čísiel, ktoré nebudeme uvádzať. Z tohto dôvodu objasníme iba, ako vytvoríme jednoduchý kryptosystém s verejným kľúčom založený na modulárnej druhej mocnine. Tento kryptosystém objavil Michael O. Rabin, preto ho nazveme RABIN.

Vytvorenie kryptosystému RABIN: Príjemca vygeneruje náhodne dve veľké prvočísla p a q, každé približne 500 ciferné. Tieto dve prvočísla sú jeho tajomstvo. Potom príjemca vypočíta číslo

$$n = p \cdot q$$

a zverejní ho spolu s funkciou $f_n(x) = x^2 \bmod n$. Takto môže hocikto zašifrovať otvorený text x vypočítaním

$$f_n(x) = x^2 \bmod n$$

a kryptotext $f_n(x)$ poslať príjemcovi.

Spôsob činnosti kryptosystému RABIN: Odosielatelia otvorené texty x zašifrujú ako $f_n(x) = x^2 \bmod n$ a pošlú príjemcovi. Bez toho, aby sme poznali p a q nie je známy efektívny algoritmus, ktorý vypočíta na základe n a $f_n(x)$ otvorený text x. Príjemca určí x na základe svojho tajomstva, hodnôt p a q. Hlavná myšlienka je takáto: Pre ľubovoľné prvočíslo p sa dá efektívne vypočítať x ako odmocnina z $f_p(x)$. Využitím teórie čísiel príjemca vie určiť odmocninu $f_n(x)$ modulo p a aj modulo q. Z týchto dvoch odmocnín vie určiť aj otvorený text x ako odmocninu $f_n(x)$ modulo $n = p \cdot q$.

Príklad 7.1 Pretože nechceme narábať s číslami, ktoré majú niekoľko sto desiatkových cifier, na znázornenie spôsobu vytvorenia kryptosystému RABIN použijeme len malé prvočísla $p = 107$ a $q = 73$. Príjemca vypočíta $n = p \cdot q = 107 \cdot 73 = 7\,811$ a zverejní

$$\text{číslo } n = 7\,811 \text{ a spôsob šifrovania } x^2 \bmod 7\,811.$$

Hocikto ho môže použiť a kladné celé čísla menšie ako 7 811 ako otvorené texty premeniť na kryptotext, ktorý pošle príjemcovi.

Uvažujme, že odosielateľ chce poslať otvorený text $x = 6\,204$. Vypočíta

$$\begin{aligned} x^2 \bmod n &= (6\,204)^2 \bmod 7811 \\ &= 38\,489\,616 \bmod 7811 \\ &= 4\,819 \\ &\quad \{\text{lebo } 38\,489\,616 = 7\,811 \cdot 4\,927 + 4\,819\ \} \end{aligned}$$

Takže kryptotext je 4 819. Pretože stále počítame modulo n, je kryptotext vždy menší ako n.

Ktoré čísla z $\{1, 2, \ldots, 7\,811\}$ umocnené na druhú modulo $n = 7\,811$ dajú číslo 4 819? Bez toho, aby sme poznali faktorizáciu $7\,811 = 107 \cdot 73$, nevieme lepší spôsob, ako vyskúšať skoro všetkých 7 811 kandidátov. V prípade čísiel n, ktorých veľkosť je až $10^{1\,000}$ je takýto postup ale fyzikálne nerealizovateľný. Keď poznáme prvočinitele n, vieme efektívne určiť všteky celé čísla y, pre ktoré rovnice

$$y^2 \bmod n = \text{kryptotext}.$$

Môžu existovať najviac štyri také hodnoty y. Ktorý z nich je otvorený text, určíme podľa významu. Alebo požiadame Alicu, aby nám poslala navyše ešte jeden bit obsahujúci špeciálnu číselno-teoretickú informáciu. □

Úloha 7.11 Vytvorte zodpovedajúci kryptosystém RABIN pre prvočísla 13 a 17. Určite kryptotext k otvorenému textu 100. Nájdite všetky y z množiny $\{1, 2, \ldots, 13 \cdot 17\}$ s vlastnosťou

$$y^2 \bmod 13 \cdot 17 = \text{kryptotext}.$$

Pretože nechceme ísť hlbšie do teórie čísiel, nepokúšame sa ani ukázať, ako vie príjemca vďaka svojmu tajomstvu - rozkladu čísla n - efektívne vypočítať otvorený text. Je ale dôležité povedať, že nemáme matematické dôkazy o tom, že predkladaní kandidáti na jednosmerné funkcie sú naozaj jednosmerné funkcie. Súvisí to s nám už známym problémom, že nie sme schopní dokázať dolné odhady zložitosti konkrétnych problémov. A tak sú všetky systémy s verejným kľúčom založené len na skúsenosti, že sa doteraz nikomu nepodarilo nájsť efektívny algoritmus na výpočet inverznej funkcie f^{-1}, teda ani na dešifrovanie. Pre predstavený kryptosystém RABIN vieme, že jeho prelomenie je rovnako ťažké ako rozklad daného čísla na prvočinitele. Presnejšie sformulované, keď vieme efektívne prelomiť kryptosystém RABIN, vieme aj efektívne rozkladať na prvočísla. A aj naopak platí, že ak existuje efektívny spôsob rozkladu na prvočinitele, potom je možné RABIN efektívne prelomiť. Voľba veľkosti prvočísiel p a q v kryptosystéme RABIN s niekoľko sto desiatkovými ciframi je potvrdená praxou, lebo ani najlepšie algoritmy na rozklad na prvočinitele

bežiace na najrýchlejších súčasných počítačoch nevypočítajú p a q pre zadané $n = p \cdot q$ ani za miliardy rokov. Preto sa ich veľkosť zväčšuje ako sa zvyšuje rýchlosť počítačov.

V čom sú prednosti systémov s verejným kľúčom? Pokúsime sa o zhrnutie:

(i) Máme iba jedno tajomstvo, ktoré je u príjemcu. Nemusí sa oň s nikým deliť, a preto sa ho nemôže ani nik od niekoho iného dozvedieť. Príjemca si ho môže sám vytvoriť.

(ii) Spôsob šifrovania sa uverejní. To je jediná komunikácia pred použitím kryptosystému s verejným kľúčom a táto komunikácia nevyžaduje šifrovanie. Po zverejnení spôsobu šifrovania môže príjemcovi poslať hocikto šifrovanú správu.

Okrem uvedených dvoch hlavných výhod kryptosystémov s verejným kľúčom môžeme objaviť veľa ďalších výhod, keď ich chceme použiť na bezpečnú komunikáciu tam, kde sa nedajú použiť symetrické kryptosystémy. Na ilustráciu ukážeme jednoduchý komunikačný protokol pre digitálny (elektronický) podpis. Po právnej stránke je vlastnoručný podpis druh záruky pravosti. V digitálnej komunikácii (napríklad pri elektronickom prevode peňazí) nemôžeme poskytnúť vlastnoručný podpis. Okrem toho by sme chceli, aby bol podpis voči falšovaniu bezpečnejší ako vlastnoručný podpis.

Sformulujme presne naše požiadavky na komunikačný protokol, ktorý chceme vytvoriť. Zákazník Z chce banke B preukázať záruku pravosti prevodného príkazu z jeho účtu, alebo podpísať nejaký iný dokument. Pritom požadujeme nasledujúce:

(i) Banka B musí byť presvedčená o pravosti podpisu Z. Aj B, aj Z musia byť chránení pred treťou stranou (falšovateľom), ktorá sa chce pred B vydávať za Z.

(ii) Z musí byť uchránený pred takými aktivitami B, pri ktorých B tvrdí, že má dokument podpísaný Z, hoci Z ho nepodpísal (B sa nesmie naučiť falšovať podpis Z).

(iii) V prípade, že Z podpísal dokument u, má B možnosť presvedčiť hocikoho tretieho, že dokument u naozaj podpísal Z.

Požiadavku (i) dokážeme splniť aj symetrickým kryptosystémom. Ale nijaký symetrický kryptosystém nemôže zaručiť súčasné splnenie podmienok (i) a (ii).

Úloha 7.12 Vytvorte komunikačný protokol pre digitálny podpis, založený na klasickom kryptosystéme, ktorý zaručí splnenie požiadavky (i).

Vlastnosť (ii) dokážeme splniť ťažšie ako (i), lebo na prvý pohľad sa javí proti našej intuícii.

> *Na jednej strane - podmienka (i) - by sa B mala presvedčiť o pravosti podpisu Z, teda očakávame, že bude vedieť niečo o spôsobe podpisovania, aby mohla podpis overiť. Na druhej strane - podmienka (ii) - B nesmie o tom, ako sa Z podpisuje, vedieť priveľa, aby ho nevedela napodobniť.*

Napriek tomu vytvoríme na základe konceptu verejného kľúča protokol spĺňajúci všetky tri požiadavky (i), (ii) a (iii).

Vytvorenie protokolu
Zákazník Z má kryptosystém s verejným kľúčom, pričom jeho verejná šifrovacia funkcia je Šif_Z a tajná dešifrovacia funkcia Dešz. Kryptosystém je komutatívny, teda pre ľubovoľný otvorený text platí:

$$\text{Dešz}(\text{Šif}_\text{Z}(\text{otvorený text})) = \text{otvorený text} = \text{Šif}_\text{Z}(\text{Dešz}(\text{otvorený text})).$$

To znamená, že môžeme nielen najprv použiť na šifrovanie Šif_Z(otvorený text) a potom na dešifrovanie $\text{Dešz}(\text{Šif}_\text{Z}$(otvorený text)), ale na šifrovanie môžeme použiť najprv aj Dešz(otvorený text) a potom na dešifrovanie

$$\text{Šif}_\text{Z}(\text{Dešz}(\text{otvorený text})) = \text{otvorený text}.$$

Banka pozná verejnú šifrovaciu funkciu Šif_Z.

Komunikačný protokol

Vstup: Dokument u, ktorý má Z podpísať.

Postup:

1. Z vezme dokument u a vypočíta $\text{Dešz}(u)$. Potom Z pošle banke B dvojicu
$$(u, \text{Dešz}(u))$$
2. Použitím verejnej šifrovacej funkcie Šif_Z na prijatý podpísaný dokument $\text{Dešz}(u)$ banka B vypočíta $\text{Šif}_\text{Z}(\text{Dešz}(u))$ a porovnaním
$$u = \text{Šif}_\text{Z}(\text{Dešz}_Z(u))$$
overí pravosť podpisu.

Splnenie podmienok korektnosti

(i) Nik iný okrem Z nevie efektívne vypočítať správu $\text{Dešz}(u)$ (podpísaný dokument u). Takže B je presvedčený o pravosti podpísaného dokumentu.

(ii) To, že B pozná u a $\text{Dešz}(u)$ jej nepomôže podpísať iný dokument u' ako $\text{Dešz}(u')$, lebo B nevie efektívne vypočítať funkciu Dešz.

(iii) Vďaka tomu, že spôsob šifrovania Šifz je verejne známy, môže banka B hocikomu, kto má o to záujem, ukázať dvojicu $(u,\text{Dešz}(u))$, a ten si môže overiť platnosť podpisu vykonaním výpočtu:

$$\text{Šifz}(\text{Dešz}(u)) = u$$

Toto elegantné riešenie problému digitálneho podpisu môže pôsobiť ozaj magicky. No to je iba začiatok mnohých ďalších zázračných protokolov, ktoré neočakávane riešia rôzne komunikačné úlohy. Odôvodnenie ich správnosti vyžaduje hlbšie vedomosti z algebry, teórie čísiel a algoritmiky, preto sa musíme, žiaľ, zriecť ich prezentácie. Sú to jedny z najkrajších príkladov vysokej a fascinujúcej užitočnosti „suchých“ výsledkov matematiky, ktorým sa často v našom školstve bohužiaľ nevenuje patričná pozornosť.

Úloha 7.13 V predstavenom protokole sa neudržiava dokument u v tajnosti, lebo sa prenáša nešifrovaný. Každý, kto načúva komunikácii, sa môže dozvedieť, čo je v u. Zmeňme teraz požiadavku (iii) takto:

(iii′) Nik tretí, kto načúva komunikácii medzi B a Z, nesmie zistiť obsah podpísaného dokumentu.

Navrhnite komunikačný protokol, ktorý spĺňa všetky tri požiadavky (i), (ii) aj (iii′).

Úloha 7.14 (tvrdý oriešok) Všimnime si problém **autentifikácie**. Tu netreba podpisovať nijaký dokument, ale musíme presvedčiť niekoho o našej identite. Požiadavky na protokol pre autentifikáciu sú nasledujúce:

(i′) rovnaká ako (i), v zmysle, že B sa presvedčila o identite Z.
(ii′) Z musí byť chránený, pred aktivitami, v ktorých by B chcela pred niekým tretím vystupovať ako Z.

Predstavený protokol nie je vhodný na autentifikáciu. B sa počas digitálnej komunikácie dozvie podpis[9] $(u, \text{Dešz}(u))$ a v komunikácii s tretím sa ním môže preukazovať ako Z. Navrhnite komunikačný protokol, ktorý spĺňa (i′) a (ii′).

[9] preukaz

Časť o kryptosystémoch s verejným kľúčom zakončíme niekoľkými poznámkami. Keď sa teraz niekomu vidí, že kryptosystémy s verejným kľúčom sú len pre jednostranú komunikáciu mnohých odosielateľov jedinému príjemcovi, neuvedomuje si ešte všetky možnosti. Môže komunikovať každý s každým. Každý, kto chce komunikovať, si vygeneruje svoje vlastné tajomstvo (napr. p a q) a zverejní vo verejnom telefónnom zozname zodpovedajúcu šifrovaciu funkciu (napr. $n = p \cdot q$). Ak chceme niekomu súkromne napísať, použijeme na zašifrovanie otvoreného textu príslušnú verejnú šifrovaciu funkciu príjemcu.

Kryptosystémy s verejným kľúčom nepredčia vo všetkých parametroch symetrické kryptosystémy. Klasické kryptosystémy, akým je DES, majú jednu podstatnú výhodu, že sú vďaka hardvérovej realizácii často stovkykrát rýchlejšie ako kryptosystémy s verejným kľúčom. To sa pri veľkých objemoch dát už významne prejaví. V praxi to vedie k tomu, že kryptosystém s verejným kľúčom sa použije iba na výmenu kľúča pre symetrický kryptosystém, ktorý sa využíva pri ďalšej komunikácii[10] na prenos všetkých ostatných údajov.

7.7 Míľniky našich objavov - cesta krajinou zázrakov kryptografie

Kryptografia sa zaoberá tvorbou kryptosystémov umožňujúcich bezpečnú výmenu tajných informácií. Pôvodne sa pestovala kryptografia ako umenie vytvárania tajných písiem. Komunikujúci (odosielateľ a príjemca) a ich protivníci (kryptoanalytici) hrali navzájom duchaplnú hru. Jedna strana vymyslela šifrovanie založené na rôznych trikoch a druhá strana sa ich snažila nájsť a odhaliť a prelomiť tým daný kryptosystém. Pojem bezpečnosti sa pri tejto hre vtipných nápadov nedal vôbec sformulovať.

Auguste Kerckhoffs vypracoval prvé požiadavky na bezpečnosť kryptosystému založené na tom, že spoľahlivosť kryptosystému závisí len od utajenia kľúča (a nie od utajenia spôsobu šifrovania). To viedlo sprvu k predstave, že na zaručenie bezpečnosti je nevyhnutný a postačujúci veľký počet kľúčov. Je to dosť zrejmá požiadavka, ale rýchlo sa ukázalo, že bezpečnosť kryptosystému nezaručí ani veľký počet kľúčov.

Na fundovanú definíciu bezpečnosti sme museli čakať, až kým sa nezačali vytvárať pojmy v informatike akými sú algoritmus, výpočtová zložitosť

[10] To znamená na väčšinu komunikácie.

a tým aj efektívna (praktická) riešiteľnosť. Potom informatika definovala bezpečnosť kryptosystému ako neexistenciu efektívneho algoritmu, ktorý by umožnil dešifrovať správu bez toho, aby poznal kľúč. Od tohto okamihu sa začínajú moderné dejiny kryptológie ako vednej disciplíny na hranici informatiky, matematiky a čoraz viac aj fyziky[11].

Klasické kryptosystémy sa vyznačujú tým, že kľúč určuje spôsob šifrovania a rovnako aj spôsob dešifrovania, tým predstavuje spoločné tajomstvo príjemcu a odosielateľa. Hlavným problémom je zhodnúť sa na jednom spoločnom kľúči skôr ako máme k dispozícii bezpečný kryptosystém. Ukázali sme si, že vieme vytvoriť komunikačný protokol na výmenu kľúča, ktorý je ale bezpečný len voči pasívnemu protivníkovi. Tajomstvo sa prezradí v prípade, keď sa protivník vie zapojiť do komunikačnej linky a vydávať sa za príjemcu.

Východisko z tejto situácie prinieslo skonštruovanie kryptosystému s verejným kľúčom. Hlavná myšlienka vychádza z teórie zložitosti a takzvaných jednosmerných funkcií. Jednosmerná funkcia sa dá efektívne vypočítať, ale zodpovedajúca inverzná funkcia nie, pokiaľ nemáme nejakú dodatočnú vedomosť (a tú má ako jediný iba príjemca). Tajná dodatočná vedomosť hrá podobnú úlohu ako svedkovia pri vytváraní efektívnych randomizovaných algoritmov. Samotná jednosmerná funkcia je zverejnená a používa sa na šifrovanie dokumentov. Tajnú vedomosť má len príjemca, ktorý s jej využitím vie pomocou inverznej funkcie efektívne dešifrovať kryptotexty.

Pretože je ťažké dokázať dolný odhad výpočtovej zložitosti konkrétnych algoritmických úloh, nik zatiaľ matematicky nedokázal, že nejaká konkrétna funkcia je jednosmerná. Vyskúšaní a v praxi používaní kandidáti na jednosmerné funkcie sú násobenie, ktorého inverzná funkcia je faktorizácia a umocňovanie (na druhú) modulo prirodzené číslo n, ktorého inverzná funkcia je počítanie zodpovedajúcej modulárnej odmocniny.

Vďaka kryptosystémom s verejným kľúčom sú tu dnešné aplikácie v oblasti e-komercie, ktoré by so symetrickým spôsobom šifrovania neboli možné. Ďalší vývoj sa uberá smerom k elektronickým voľbám a mnohým iným aplikáciám.

Koncept kryptosystémov bol prvý raz predstavený v roku 1976 dvojicou Diffie a Hellman [DH76]. Najrozšírenejší systém je RSA kryptosystém, ktorý vymysleli v roku 1978 Rivest, Shamir a Adleman [RSA78]. Podobne

[11] Kvôli kvantovým efektom používaným v kryptografii. Viac o tejto téme prezentujeme v kapitole 9.

ako v mnohých iných prípadoch trvalo približne 20 rokov, kým sa tento neuveriteľný poznatok základného výskumu dočkal širokého komerčného využitia.

Ako úvod do kryptografie odporúčame knihy [Beu02a, Beu02b]. Záujemcom o hlbšie vedomosti doporučujeme knihy Salomaa [Sal96], Delfs und Knebl [DK02] a [Hro04c].

Návody na riešenie vybraných úloh

Úloha 7.5

a) Máme:

$$\begin{array}{rr} & 00110011 \\ \perp & 00101101 \\ \hline & 11100001 \\ \perp & 00101101 \\ \hline & 00110011 \end{array}$$

b) Tento spôsob funguje, pretože platí

$$(a \perp c) \perp c = a,$$

a teda dvojnásobnou aplikáciou kľúča na otvorený text dostaneme opäť ten istý otvorený text.

c) Výsledok operácie $\perp$ je vždy obrátením výsledku operácie $\oplus$. Keď platí $a \oplus b = 1$, potom platí $a \perp b = 0$ a v prípade, že platí $a \oplus b = 0$, platí aj, že $a \perp b = 1$.

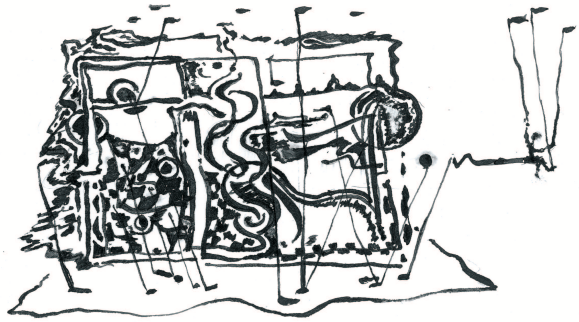
Úloha 7.6 Osoba A vygeneruje náhodne postupnosť bitov $a_1a_2 \ldots a_n$ a použije operáciu $\oplus$ na to, aby zašifrovala otvorený text $k_1k_2 \ldots k_n$ na kryptotext $k_1 \ldots k_n \oplus a_1 \ldots a_n$. Potom A znova vygeneruje n bitov $b_1b_2 \ldots b_n$ a vypočíta:

$$\begin{array}{rr} & a_1a_2 \ldots a_n \\ \oplus & b_1b_2 \ldots b_n \\ \hline & c_1c_2 \ldots c_n \ . \end{array}$$

Potom A pošle kľúč $b_1b_2 \ldots b_n$ osobe B a kľúč $c_1c_2 \ldots c_n$ osobe C. Pretože je postupnosť bitov $b_1b_2 \ldots b_n$ zvolená náhodne, ani B ani C nemôžu určiť kľúč $a_1a_2 \ldots a_n$, s ktorým bol zašifrovaný otvorený text $k_1k_2 \ldots k_n$. Bez toho, aby poznali kľúč $a_1a_2 \ldots a_n$ nie je možné dešifrovať kryptotext $d_1d_2 \ldots d_n$. Ak ale B a C vzájomne spolupracujú, dokážu kľúč $a_1a_2 \ldots a_n$ vypočítať nasledovne:

$$\begin{array}{rr} & b_1b_2 \ldots b_n \\ \oplus & c_1c_2 \ldots c_n \\ \hline & a_1a_2 \ldots a_n \ . \end{array}$$

Keď majú pôvodný kľúč $a_1a_2 \ldots a_n$, vedia spočítať otvorený text $k_1k_2, \ldots k_n = d_1d_2 \ldots d_n \oplus a_1a_2 \ldots a_n$.



Vedecké objavy sa robia takto:
Všetci vedia, že sa niečo nedá urobiť.
Potom príde niekto, kto to nevie
a objaví to.

Albert Einstein

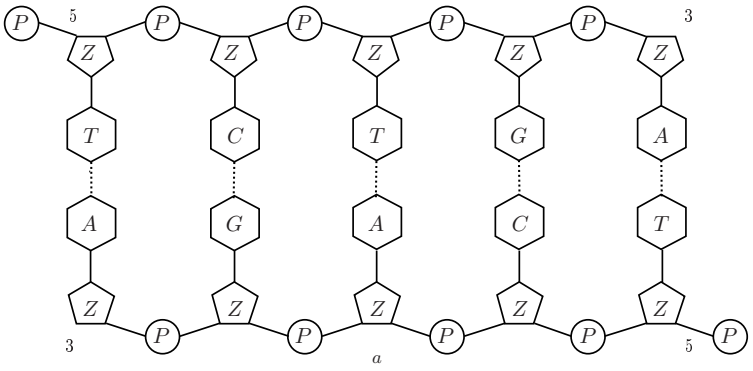
Kapitola 8

Počítanie s DNA molekulami alebo biopočítačová technológia na obzore

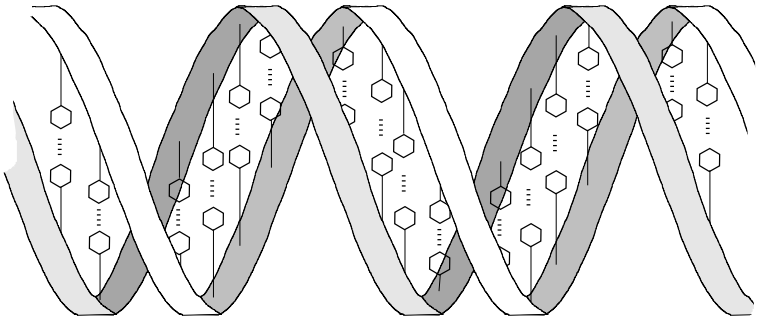
8.1 Ako to bolo doteraz

Mnohé science fiction romány sa začínajú zmesou obsahujúcou biologické a elektronické prísady alebo spájajú ľudský mozog s počítačovými komponentmi a na konci je z toho inteligentný biorobot. Téma tejto kapitoly je vzdialená od takýchto utopických predstáv a nerealistických proroctiev niektorých vedcov v oblasti umelej inteligencie zo šesťdesiatych rokov. Predstavíme už existujúcu technológiu biopočítačov, ktorá nie je len hypotetická. Či sa ona presadí v praxi, závisí od toho, ako pokročí v ďalšom období rozvoj biochemických metód na výskum DNA sekvencií.

Ako takéto biopočítače vyzerajú? Ako sme sa dostali k realistickej biologickej počítačovej technológii? Naša skúsenosť s počítačmi za posledných päťdesiat rokov ukazuje, že sú každé dva roky dvakrát menšie a súčasne dvakrát rýchlejšie ako predtým. Exponenciálne zlepšenie v čase. Takto



obr. 8.1



obr. 8.2

to nemôže pokračovať donekonečna, elektronické technológie čoskoro dosiahnu hranice miniaturizácie, čím sa definitívne skončí rýchly rast výkonu počítačov. Fyzikálne hranice výkonu počítačov sa spočítali už veľmi dávno, a už v roku 1959 sa známy fyzik Richard Feyman pýtal: „Ako to pôjde ďalej? Môžeme miniaturizovať tým, že realizujeme výpočtové procesy na úrovni molekúl a častíc?“ Výsledkami týchto úvah sú DNA-počítač a kvantový počítač, ktoré predstavíme v tejto a nasledujúcej kapitole.

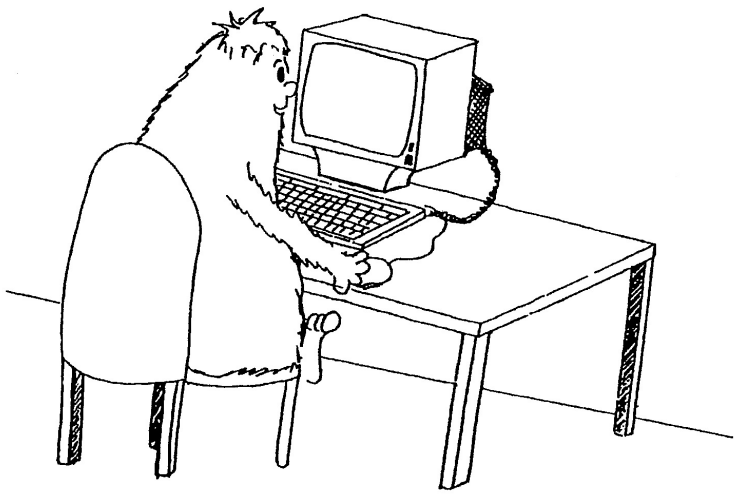
Teraz rozumieme, že potreba miniaturizácie a s ňou spojený nárast výkonu viedol k želaniu počítať na úrovni molekúl a atómov. Je ale toto

želanie realizovateľné? Nie je to neprirodzená myšlienka? Máme nútiť molekuly robiť niečo, na čo vôbec neboli určené? Prv ako odpovieme na tieto otázky, zamyslime sa nad tým, čo je prirodzené, a čo neprirodzené. V našom umelom svete matematických modelov skutočnosti pracujeme so symbolmi. Všetky čísla alebo iné údaje sú zapísané ako texty, teda postupnosti symbolov. Už sme sa naučili, že prácu počítača si môžeme vo všeobecnosti predstaviť ako transformovanie vstupných textov (údajov) na výstupné texty (údaje).

Ako je to s DNA sekvenciami? Vieme, že sú nositeľmi biologických informácií a že všetky procesy v živých organizmoch sú riadené informáciami uloženými v DNA sekvenciách. V súčasnosti rozumieme len malému zlomku tohto riadenia, ale vôbec nepochybujeme, že na biologické procesy môžeme nazerať aj ako na spracovávanie informácií. Napriek tomu nie sme dostatočne ďaleko v chápaní týchto procesov, aby sme ich vedeli využiť na počítanie. Hlavná myšlienka DNA počítača je oveľa jednoduchšia. DNA sekvencie si môžeme predstaviť ako texty zostavené z písmen `A`, `C`, `G` a `T`. Štyri písmená predstavujú bázy Adenin (`A`), Cytosin (`C`), Guanin (`G`) a Thymin (`T`), z ktorých je zložený DNA kód. Typicky sa DNA vyskytuje ako molekula v tvare dvojitého reťazca (obr. 8.1 a obr. 8.2), pričom väzby vznikajú len medzi `A` a `T` a medzi `G` a `C`. Chemické väzby $\texttt{A} - \texttt{T}$ a $\texttt{G} - \texttt{C}$ sú podstatne slabšie ako ostatné väzby v reťazci na obr. 8.1. Dôležité je, že do určitej miery vieme regulovať molekulárnu stabilitu DNA. Naše údaje si môžeme predstaviť ako texty zložené so symbolov `A`, `C`, `G` a `T` a zo zodpovedajúcej symbolickej DNA sekvencie k nim vytvoriť fyzickú reprezentáciu. S údajmi reprezentovanými týmto spôsobom môžeme vykonávať v reagenčných nádobách v laboratóriu biochemické operácie, ktorými DNA sekvencie meníme. Vytvorené DNA sekvencie na konci prečítame a interpretujeme ako výsledok.

Na počudovanie sa dá matematicky dokázať, že takéto DNA počítače sú schopné urobiť presne to isté ako klasické elektronické počítače. To znamená, že naše chápanie algoritmickej riešiteľnosti sa ani trochu nespochybnilo. Čo dokážeme vyriešiť algoritmicky (automaticky na počítači), to vieme urobiť aj s DNA algoritmami a platí to aj opačne.

Čo máme z toho, keď namiesto elektronického počítača použijeme DNA počítač? Kvapka vody obsahuje 10^{19} molekúl. Keď máme v reagenčnej miske 10^{21} DNA sekvencií, všetky operácie sa vykonajú súčasne (paralelne) na všetkých 10^{21} molekulách. Na bežnom počítači nikdy nevykonáte súčasne operáciu s 10^{21} údajmi. Tadiaľ vedie cesta k ďalšiemu želanému urýchleniu výpočtových procesov.

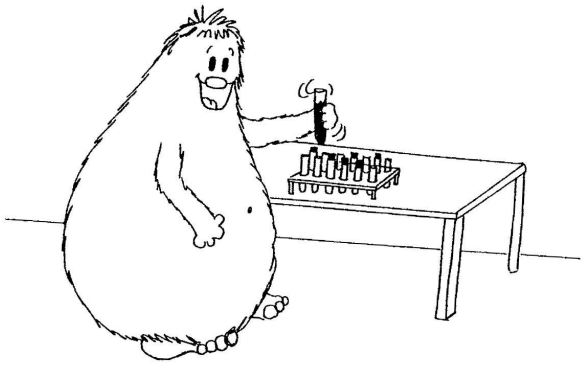


obr. 8.3

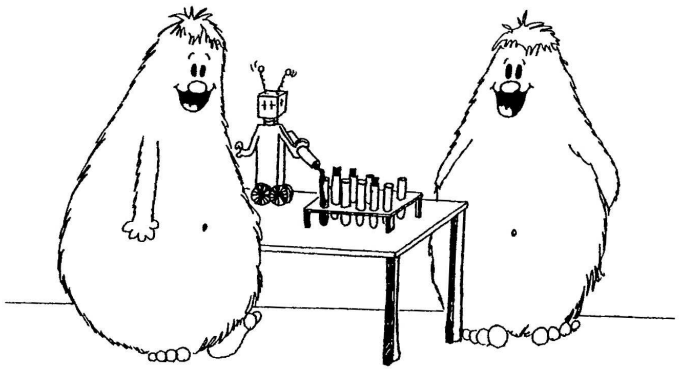
Preto sme dnes azda na začiatku produktívnej konkurencie dvoch rôznych počítačových technológií. Táto predstava je zobrazená na obrázkoch (obr. 8.3, obr. 8.4 a obr. 8.5) v knihe „DNA-Computing“, ktorú napísali Păun, Rozenberg a Salomaa. Na obr. 8.3 vidíme dnes vedúcu a najrozšírenejšiu technológiu klasických elektronických počítačov. Obr. 8.4 predstavuje DNA-Computing, pri ktorom sa vykonávajú biochemické operácie v reagenčných nádobách v laboratóriu. Čo vznikne z tejto konkurencie? Možno rozumná zmes elektroniky a biomasy. Namiesto toho, aby sa chemické operácie vykonávali ručne, môže ich realizovať elektronický robot ako na obr. 8.5.

V tejto kapitole ukážeme v odseku 8.2 zoznam realizovateľných biochemických operácií, ktoré postačujú na to, aby sme mohli vykonať ľubovoľný výpočet. Potom prezentujeme známy experiment Adlemana[1], ktorý ako prvý vytvoril „biopočítač“ a vo svojom laboratóriu na ňom na začiatku deväťdesiatych rokov vyriešil konkrétny prípad optimalizačnej úlohy obchodného cestujúceho. Nakoniec všetko zhrnieme a prediskutujeme silné a slabé stránky technológie DNA počítačov a objasníme, za akých predpokladov ich môže čakať úspešná budúcnosť.

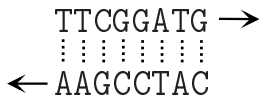
[1]Všimnite si, že je to ten istý Adleman, ktorý je jedným z troch vynálezcov známeho RSA kryptosystému.



obr. 8.4



obr. 8.5



obr. 8.6

8.2 Ako sa dá premeniť laboratórium na biopočítač

Pri varení podľa receptu a pri modelovaní počítača sme sa v druhej kapitole naučili, že keď sa chceme dohodnúť na pojmoch „počítač“ a „algoritmus“, musíme zafixovať spôsob reprezentovania a pamätania si údajov a tiež určiť zoznam operácií s údajmi, pri ktorých nemáme pochybnosti ako ich vykonať.

V modeli DNA počítača sú údaje DNA sekvencie ako molekuly v tvare dvojitého reťazca. Tieto dvojreťazcové molekuly sú fyzickým nosičom informácie. Samotné DNA molekuly môžeme uložiť v reagenčných nádobách. K dispozícii máme konečný počet reagenčných nádob.

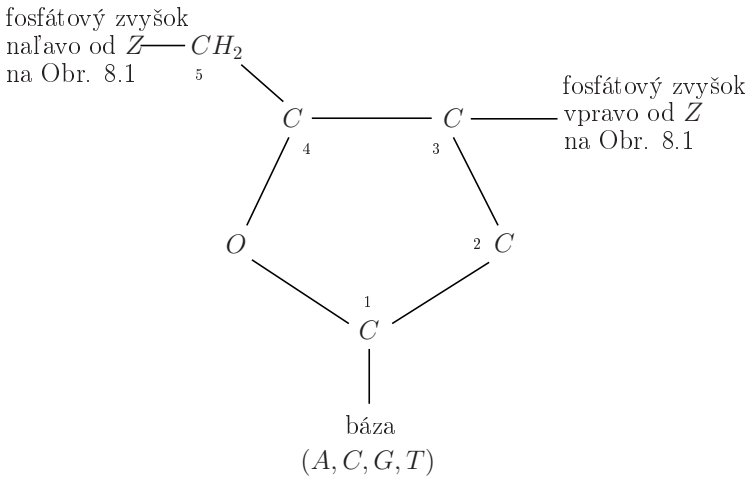
Aby sme mohli s obsahmi reagenčných nádob vykonať operácie, môžeme používať rôzne prístroje a potrebné predmety. Nemáme v úmysle podať detailné vysvetlenie, ako a prečo sa dajú vykonať príslušné biochemické operácie, pretože nepredpokladáme rozsiahle predchádzajúce vedomosti z molekulárnej biológie. Na druhej strane chceme sprostredkovať aspoň predstavu o tom, z akých dôvodov sú možné určité chemické operácie. Preto si stručne zopakujeme niektoré základné vedomosti z biológie[2].

James D. Watson a Francis H. C. Crick v roku 1953 objavili, že DNA molekula ma štruktúru v tvare dvojitej závitnice (obr. 8.1, obr. 8.2). Nobelova cena, ktorú za to dostali, je len malým potvrdením skutočnosti, že to bol jeden z najvýznamnejších objavov dvadsiateho storočia. Fakt, že sa navzájom môžu spájať len bázy `G` s `C` a `A` s `T`, nazývame **Watson-Cricková komplementárnosť**. Idealizovaná predstava[3] DNA molekuly je dvojitý reťazec na obr. 8.6.

Úloha 8.1 Nakreslite alebo doplňte kresbu molekúl DNA, ktorých vrchné reťazce sú `AACGTAT`, `GCCACTA` a `AACG`.

[2] Podrobné predstavenie pre nebiológov môžeme nájsť v [BB03, PRS05].

[3] Molekuly DNA majú zložitú trojrozmernú štruktúru, ktorá je podstatná z hľadiska ich funkčnosti.



obr. 8.7

Väzby v reťazci

TTCGGATG

sú približne desaťkrát silnejšie ako chemické väzby medzi nukleotidmi A · · · T a G · · · C. Reťazce

TTCGGATG

a

AAGCCTAC

v dvojitom reťazci (obr. 8.6) majú svoj smer, Vrchný ide zľava doprava a spodný ide zprava doľava. Smer dostaneme, keď očíslujeme atómy uhlíka v cukre (Z na obr. 8.1), ktoré vytvárajú väzby s fosfátovými zvyškami (P na obr. 8.1) a s bázou (jednou z báz A, T, C alebo G na obr. 8.1). Je presne 5 uhlíkových atómov na jeden nukleotid[4] a zjednodušene môžeme situáciu znázorniť ako na obr. 8.7.

Na obr. 8.7 je znázornená molekula cukru (Z na obr. 8.1). Päť atómov uhlíka C je očíslovaných $1', 2', 3', 4'$ až $5'$. Atóm $1'$ sa stará o väzbu k báze. Atóm uhlíka $3'$ zodpovedá za väzbu na fosfátový zvyšok v reťazci vpravo a uhlíkový atóm $5'$ tvorí väzbu s fosfátovým zvyškom v reťazci vľavo. Preto sa smer zľava doprava označuje v biológii ako smer $5' \to 3'$.

Pre nás je teraz dôležité vedieť len to, že zvýšením energie (napr. ohriatím) môžeme dosiahnuť rozpad väzieb medzi bázami A · · · T a C · · · G, ktoré

[4]Nukleotid sa skladá z fosfátového zvyšku, jedného cukru a jednej zo štyroch báz A, C, G, T.

sú podstatne slabšie ako väzby v rámci jednotlivých reťazcov. Tak vieme rozdeliť dvojreťazcovú DNA molekulu na dve samostatné jednoreťazcové DNA molekuly. Za „vhodných“ podmienok sa môže z dvoch samostatných reťazcov znova vytvoriť jedna dvojreťazcová molekula, ale iba, keď sú postupnosti báz Watson-Crick komplementárne.

Ďalšou dôležitou vlastnosťou molekuly DNA je, že má záporný náboj, ktorého veľkosť je priamo úmerná dĺžke molekuly.

Teraz už vieme dosť na to, aby sme si mohli predstaviť niektoré základné chemické operácie s obsahmi reagenčných misiek. Z týchto operácií potom môžeme zostavovať DNA-programy. Označme si reagenčné nádoby ako R_1, R_2, R_3 atď.[5]

(i) **Union**(R_i, R_j, R_k)
`Obsahy reagenčných nádob` R_i `a` R_j `daj do reagenčnej nádoby` R_k`.`

(ii) **Amplify**(R_i)
`Znásob počet DNA-postupností v` R_i`.`

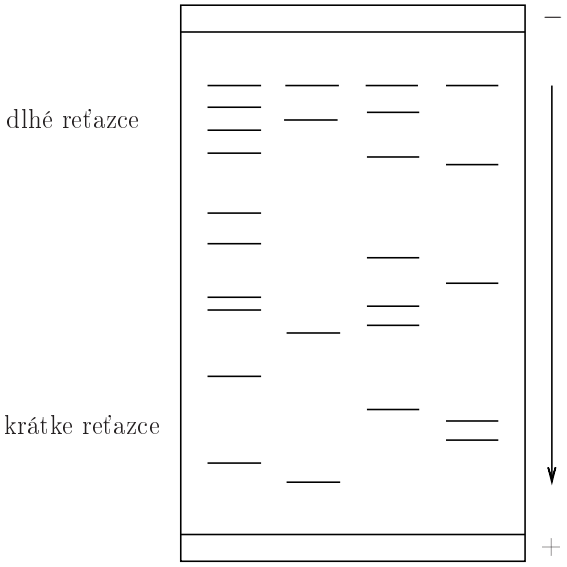
Tieto operácie sú založené na Watson-Crickovej komplementárnosti a nazývajú sa polymerázová reťazová reakcia. Táto metóda spôsobila revolúciu v molekulárnej biológii a Kary Mullis, ktorý ju v roku 1985 objavil, dostal za tento objav Nobelovu cenu. Dvojité DNA molekuly sa zahrievajú dovtedy, pokým sa nerozpoja väzby medzi bázami. Tým dostaneme z dvojitého reťazca dva jednoduché, bez toho, aby sa tieto porušili. Táto fáza sa nazýva **denaturovanie**. Potom sa do vzniknutej „DNA polievky“ pridajú[6] jednotlivé nukleotidy a všetko sa ochladí. Pritom sa nukleotidy naviažu na zodpovedajúce komplementárne bázy jednotlivých jednoduchých reťazcov a znovu sa vytvoria identické dvojreťazcové DNA-molekuly, čím sa ich počet zdvojnásobí. Celý proces sa niekoľkokrát opakuje.

(iii) **Empty?**(R_i)
`Testuj, či` R_i `obsahuje aspoň jednu DNA-molekulu alebo vôbec nič.`

(iv) **Length-Separate**(R_i, l) pre $l \in \mathbb{N}$.
`Táto operácia odstráni z` R_i `všetky DNA-sekvencie, ktoré nemajú dĺžku presne` l `báz`

[5] podobne ako registre v počítači

[6] Toto je značne zjednodušená predstava ďalších fáz polymerázovej reťazovej reakcie nazývajúcich sa „Priming“ a „Extension“. Podrobnejšie informácie možno nájsť v [PRS05].



obr. 8.8

Pri týchto operáciách sa využíva technika gélovej elektroforézy. Vieme, že keď umiestnime DNA-molekuly do elektrického poľa, budú sa vďaka ich zápornému náboju premiestňovať smerom ku kladnej elektróde. Veľkosť molekuly ju brzdí a znižuje jej rýchlosť. Zároveň čím má molekula väčší náboj, tým rýchlejšie sa pohybuje. Väčšie molekuly majú väčší náboj. Výsledkom je, že sa brzdiaci a zrýchľujúci faktor navzájom zrušia a všetky molekuly sa pohybujú ku kladnej elektróde rovnakou rýchlosťou. Preto pridáme do poľa gél, ktorý dodatočne zníži rýchlosť pohybu väčších molekúl (dlhších DNA-sekvencií). Dôsledkom toho sa kratšie DNA-molekuly pohybujú rýchlejšie ako dlhé (obr. 8.8). Dá sa vypočítať rýchlosť jednotlivých DNA-molekúl v závislosti od ich dĺžky. Keď dosiahne prvá (najkratšia) molekula kladnú elektródu, elektrické pole sa vypne a podľa dĺžok prejdených dráh sa dajú určiť dĺžky DNA reťazcov. Keďže DNA-molekuly sú bezfarebné, aby sme mohli celý proces sledovať, môžeme ich označiť fluorescenčnými farbami.

(v) **Concatenate**(R_i)
`DNA-sekvencie v` R_i `sa môžu náhodne spájať`[7] `do dlhších, dlhšie DNA-sekvencie vznikajú tým, že sa kratšie zapoja za seba.`

[7] Presnejšie vysvetlenie je v časti 8.3.

(vi) **Separate**(R_i, w) pre reagenčnú nádobu R_i a DNA sekvenciu w.
`Operáciou sa odstránia z` R_i `všetky DNA-molekuly, ktoré v sebe neobsahujú sekvenciu` w.

Napríklad $w =$ `ATTC` je časťou $x =$ `AATTCGATC`, lebo sa v celku vyskytuje v x. Vykonanie takejto operácie vyžaduje trochu viac námahy. Najprv môžeme všetky dvojité reťazce zahriatím rozpojiť na jednoduché. Potom pridať mnoho kópií DNA sekvencií komplementárnych k w a mierne všetko ochladiť. Napríklad k reťazcu $w =$ `ATTC` je komplementárny reťazec `TAAG`. Sekvencie komplementárne k w sa naviažu na zodpovedajúce časti w jednoduchých reťazcov. Jednoduché reťazce, ktoré neobsahujú w, sa nedoplnia a ostanú jednoduché. Potom všetko prefiltrujeme cez filter prepúšťajúci len jednoduché reťazce. To, čo ostane, sú reťazce ako

```
AATTCGATC
 ⋮⋮⋮⋮
 TAAG
```
,

kde nie sú už jednoduché reťazce, ale ani kompletné dvojité reťazce. No pridaním jednotlivých nukleotidov môžeme úplný dvojitý reťazec opäť reprodukovať.

(vii) **Separate-Prefix**(R_i, w) pre R_i a DNA-sekvenciu w.
`Odstránime všetky DNA-molekuly, ktoré sa nezačínajú`[8] `na` w.

(viii) **Separate-Suffix**(R_i, u) pre R_i a DNA-postupnosť.
`Odstráň všetky DNA-molekuly, ktoré sa nekončia na` u.

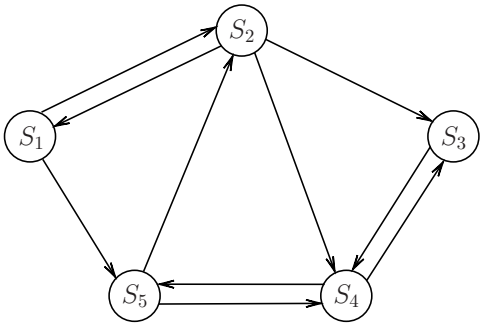
Úloha 8.2 Nech reagenčná nádoba R obsahuje `ATTGCCATGCC`, `ATATCAGCT`, `TTGCACGG`, `AACT`, `AGCATGCT`.

Ktoré DNA-molekuly zostanú po vykonaní nasledujúcich operácií?

a) Length-Separate $(R, 7)$
b) Separate $(R,$ `TTGC`$)$
c) Separate-Prefix $(R,$ `TTGC`$)$
d) Separate-Suffix $(R,$ `GCT`$)$

Úloha 8.3 Chcete vykonať operáciu Separate$(R,$ `AACT`$)$. Ktoré DNA-reťazce musíte po zahriatí pridať k R, aby ste mohli nasledujúcou filtráciou oddeliť nevhodné jednoduché reťazce?

[8] Tu sa zriekneme podrobnejšej predstavy, ako sa to uskutočňuje.



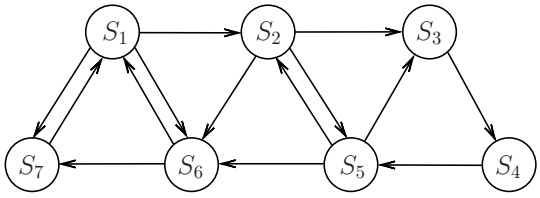
obr. 8.9

Z uvedených 8 operácii sa už dá vytvoriť DNA počítač, ktorý vie robiť to isté ako PC. Ako sa to dá, ukážeme v nasledujúcom odseku.

8.3 Adlemanov experiment alebo biohľadanie cesty

V časti 8.2 sme tvrdili, že s tam uvedenými biochemickými operáciami vieme urobiť všetko, čo sme schopní vykonať aj na klasickom počítači. Nechceme tu robiť matematický dôkaz, ale je najvyšší čas uviesť aspoň príklad ako DNA počítač vie vyriešiť algoritmický problém. Všimneme si problém Hamiltonovej cesty v orientovanom grafe (v literatúre sa označuje ako **HPP**) a povieme si, ako Adleman v chemickom laboratóriu úspešne vyriešil prípad problému HPP. Adlemanov experiment sa ukázal ako velmi podnetný a mnohé špičkové univerzity realizujú výskumné projekty, v ktorých sa usilujú vyvíjať DNA technológiu na riešenie ťažkých problémov.

Prípad HPP problému zadáme prostredníctvom cestnej siete (alebo siete leteckých liniek). Mestá (alebo križovatky) sú znázornené (obr. 8.9) ako vrcholy a spojenia medzi mestami ako rovné čiary. Keď sa čiary pretínajú mimo mesta, nemá to pre nás význam (rovnako ako pri leteckých linkách), pretože sa nemôžeme z jednej dostať na druhú. Spojnice sú jednosmerné. Keď použijeme spojnicu $Lin(s_1, s_2)$ z s_1 do s_2, dostaneme sa nevyhnutne do s_2. Ďalšia časť prípadu sú mená dvoch rôznych miest s_i a s_j. HPP problém je rozhodovací problém a otázka je, či je možné začať v s_i, prejsť práve raz každým mestom v sieti a skončiť v s_j. Takáto cesta sa nazýva **Hamiltonova cesta** z s_i do s_j.



obr. 8.10

Znázornime si to na konkrétnom prípade problému. Prípadom je sieť na obr. 8.9, štart je v s_1 a cieľ v s_5. Riešením je napríklad

$$s_1 \rightarrow s_2 \rightarrow s_3 \rightarrow s_4 \rightarrow s_5$$

(Hamiltonova cesta z s_1 do s_5), lebo každé mesto sme navštívili práve raz a navštívili sme všetky mestá v sieti. Na prejdenie cesty potrebujeme štyri spojnice $Lin(s_1, s_2)$, $Lin(s_2, s_3), Lin(s_3, s_4)$ a $Lin(s_4, s_5)$, ktoré v sieti aj skutočne sú. Takže správna odpoveď pre túto inštanciu problému je „ÁNO“.

Pre sieť na obr. 8.9, štart s_2 a cieľ s_1 neexistuje Hamiltonova cesta. Správna odpoveď v tomto prípade musí byť „NIE“. Spoznáme to podľa toho, že cieľ s_1 je prístupný len zo štartu s_2 a to jediným spôsobom. Z s_2 musíme ale najskôr navštíviť všetky ostatné mestá. Keď ale potom chceme ísť do s_1, musíme prejsť opäť cez s_2, čo ale nie je dovolené, lebo by sme ho navštívili druhý raz.

Úloha 8.4 Pozrime si sieť na obr. 8.10. Existuje Hamiltonova cesta

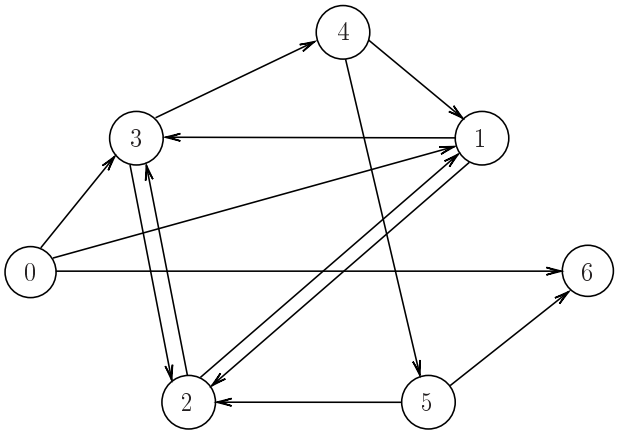
a) z s_1 do s_7?
b) z s_7 do s_1?
c) z s_4 do s_3?
d) z s_5 do s_1?

Úloha 8.5 Pre ktoré dvojice miest štart a cieľ existujú Hamiltonové cesty v sieti

a) na obr. 8.9 ?
b) na obr. 8.10 ?
c) na obr. 8.11 ?

V ďalšom budeme nazývať **cesta** z s_1 do s_n, ľubovoľnú postupnosť uzlov s_1, s_2, ..., s_n, takých, že v sieti sú spojenia

$$Lin(s_1, s_2), Lin(s_2, s_3), \ldots, Lin(s_{n-1}, s_n).$$



obr. 8.11

Namiesto $Lin(s_i, s_j)$ budeme skrátene písať $e_{i\to j}$. Teda s_1, s_7, s_1, s_7, s_1 alebo $s_7, s_1, s_2, s_5, s_2, s_5, s_6, s_1, s_7$ sú cesty, lebo na ceste sa môžu uzly ľubovoľne opakovať.

Adleman uvažoval nad prípadom problému na obr. 8.11 so štartom s_0 a s cieľom s_6. Jeho stratégia bola nasledujúca:

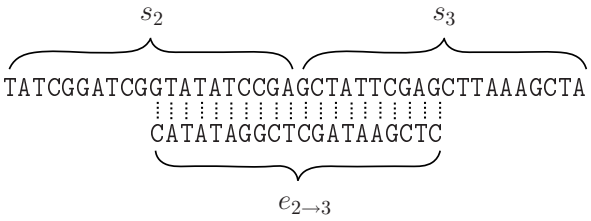
> *Mená miest v sieti zakóduj sekvenciami DNA. Potom umožni, aby sa mená vzájomne prepojených miest mohli spojiť za sebou do DNA sekvencie. Použi toľko DNA sekvencií pre každé miesto v sieti, že pri náhodne zvolených vzájomných prepojeniach vzniknú všetky možné cesty v sieti. Aplikáciou rôznych operácií Separate odstráň z reagenčnej nádoby všetky cesty, ktoré nepredstavujú Hamiltonovu cestu z s_0 do s_6.*

Prv ako biochemicky uskutočnil túto stratégiu, vybral pre miesta najprv „DNA mená“ ako texty z písmen `A`, `C`, `G`, `T`.

Napríklad nasledujúce sekvencie báz dĺžky 20 zvolil ako jednoduché reťazce predstavujúce s_2, s_3 a s_4:

$$\begin{aligned} s_2 &= \texttt{TATCGGATCGGTATATCCGA} \\ s_3 &= \texttt{GCTATTCGAGCTTAAAGCTA} \\ s_4 &= \texttt{GGCTAGGTACCAGCATGCTT} \end{aligned}$$

Teraz chceme reprezentovať prepojenie (ulicu) z s_i do s_j ako také jednoduché DNA reťazce, že operáciou Concatenate(R) budú môcť vzniknúť iba také dlhšie DNA reťazce, ktoré zodpovedajú existujúcej ceste v sieti.



obr. 8.12

Pritom využijeme vlastnosti DNA, že bázy môžu vytvárať výlučne dvojice `A` s `T` alebo `C` s `G`. Pre cestu z $e_{i\to j}$ z s_i do s_j

- rozdelíme ich texty (DNA reprezentácie s_i a s_j) vždy v strede na dve polovice;
- vytvoríme takzvané doplnky k druhému dielu s_i a k prvému dielu s_j: Nahradíme `A` s `T`, `C` s `G` a tiež aj obrátene;
- pre ulicu $e_{i\to j}$ vytvoríme text (DNA reprezentáciu) spojením týchto dvoch doplnkov.

Všimnime si, že sme tým určili aj smer ulice. V našom príklade to znamená

$e_{2\to3}$ = `CATATAGGCT CGATAAGCTC`;

$e_{3\to2}$ = `GAATTTCGAT ATAGCCTAGC`;

$e_{3\to4}$ = `GAATTTCGAT CCGATCCATG`.

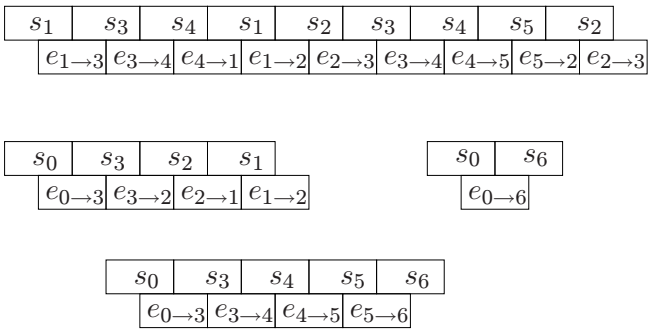
Tým sa jednotlivé reťazce pre s_2 a s_3 môžu spojiť do jedného reťazca pre $e_{2\to3}$ ako na obr. 8.12.

Úloha 8.6 Nakreslite prepojenie s_3 a s_4 spojom $e_{3\to4}$ analogicky ako je to pre s_2 a s_3 na obr. 8.12.

Úloha 8.7 Predstavme si, že do siete ulíc na obr. 8.11 pridáme spojenie $e_{2\to4}$. Aký musíme zobrať pre $e_{2\to4}$ reťazec? Nakreslite zodpovedajúce prepojenie s_2 a s_4.

Keď teraz zoberieme jednoduché reťazce ako DNA sekvencie zodpovedajúce tomuto kódovaniu a dáme ich za vhodných podmienok do reagenčnej nádoby, môžu sa spájať do dvojitých reťazcov tak, ako je to na obr. 8.13.

Každý takýto dvojitý reťazec opisuje nejakú cestu v cestnej sieti. Aby to celé bezchybne fungovalo, musíme zabezpečiť, aby sa zakódovania ciest



obr. 8.13

mohli predlžovať len spôsobom, ako je to v uvedenom príklade. Predovšetkým musia byť všetky polovice kódov miest navzájom rôzne.

Po tom, čo sme mestá a cesty takto zakódovali, môžeme zrealizovať Adlemanovu stratégiu hľadania Hamiltonovej cesty nasledujúcim DNA algoritmom pre danú sieť n miest $s_0, s_1, \ldots, s_{n-1}$:

1. Daj do reagenčnej nádoby T DNA kódy všetkých miest a ciest (ako jednoduché reťazce). Pridaj DNA sekvenciu dĺžky 10^9, ktorá je doplnkom k prvej polovici s_0 a DNA sekvenciu dĺžky 10, ktorá je doplnkom k druhej polovici cieľového miesta s_{n-1}.

2. Opakuj $(2n \cdot \log_2 n)$-krát operáciu Amplify(T), aby vzniklo aspoň n^{2n} kópií z každého z týchto DNA reťazcov.

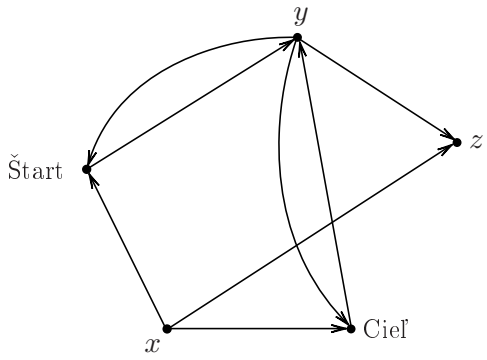
3. S Concatenate(T) vytvor veľké množstvo dvojreťazcových DNA sekvencií, ktoré predstavujú rôzne dlhé cesty v cestnej sieti. Tento proces prebieha náhodne a veľký počet mien miest nám zaručí, že s vysokou pravdepodobnosťou[10] vznikne každá zo všetkých možných ciest, ktorých dĺžka je až n.

4. Aplikuj operáciu Length-Separate(T, l), pričom l je n násobok dĺžky kódu jedného mesta. V T ostanú len kódovania zodpovedajúce cestám, ktoré prechádzajú presne cez n miest.

5. Použi Separate-Prefix(T, s_0), aby v T ostali len také DNA sekvencie, ktoré zodpovedajú kódom ciest začínajúcim sa v s_0 = ŠTART.

[9] Dĺžka 10 zodpovedá našej sieti z obr. 8.11. Pre kódy miest v iných sieťach sú možné iné dĺžky.

[10] Prirodzene pri tom náhodne vzniknú aj cesty dlhšie ako n.



obr. 8.14

6. Použi Separate-Suffix(T, s_{n-1}), aby v T ostali len také DNA sekvencie, ktoré zodpovedajú kódom ciest končiacim sa v s_{n-1} = CIEĽ.

7. Aplikuj postupnosť $(n-2)$ operácií

$$Separate(T, s_1), Separate(T, s_1), \ldots, Separate(T, s_{n-2}).$$

Po vykonaní postupnosti operácií v T ostanú len tie cesty, ktoré každé mesto z $\{s_0, s_1, \ldots, s_{n-1}\}$ obsahujú aspoň raz. Pretože vykonanie kroku 4 nám zaručí, že v T sú len cesty prechádzajúce presne n mestami, obsahujú tieto cesty každé mesto práve raz.

8. Preskúmaj obsah T s Empty?(T) a odpovedaj ÁNO, ak v T ostala aspoň jedna DNA sekvencia. V opačnom prípade odpovedaj NIE.

Použitím algoritmu na sieť na obr.8.11 so štartom s_0 a cieľom s_6 dosiahneme, že v reagenčnej nádobe ostanú molekuly dvojitého reťazca DNA, predstavujúce Hamiltonovu cestu

$$s_0 \to s_1 \to s_2 \to s_3 \to s_4 \to s_5 \to s_6.$$

Teda potvrdili sme správnu odpoveď ÁNO.

Úloha 8.8 Nájdite aspoň tri rozličné cesty v sieti z obr. 8.11, ktoré ostanú v reagenčnej nádobe po vykonaní piatej operácie z Adlemanovho DNA algoritmu. Čo zostane po vykonaní šiestej operácie?

Úloha 8.9 Pozrime si cestnú sieť na obr. 8.14:

a) Uveďte (čo najkratšie) kódovanie (prostredníctvom DNA) miest, také, aby boli splnené nasledujúce podmienky:

1. Kód každého mesta sa odlišuje od kódu ľubovolného iného mesta najmenej na štyroch pozíciách.
2. Prvá i druhá polovica kódu každého mesta sa odlišuje od prvej aj druhej polovice kódu ľubovoľného iného mesta.

b) Určte DNA kódy ulíc, ktoré sa budú hodiť k vašim kódom miest.

Úloha 8.10 Opíšte detailne priebeh Adlemanovho algoritmu pre vstup z úlohy 8.9, pričom uvediete postupnosť operácií s konkrétnymi parametrami (DNA sekvenciami a dĺžkami).

Úloha 8.11 Sú požiadavky na kódovanie miest uvedené v úlohe 8.9 (a) postačujúce na to, aby každý vytvorený dvojitý DNA reťazec zodpovedal ceste ulicami? Opodstatnite vašu odpoveď.

Vidíme, že Adlemanov algoritmus z pohľadu algoritmiky používa veľmi jednoduchú stratégiu. Využíva masívny paralelizmus a vytvára všetky možné cesty sieťou, ktoré sú kandidátmi na riešenie. Potom skúša vytriediť spomedzi kandidátov hľadané riešenie (Hamiltonovu cestu od štartu k cieľu).

Od začiatku 90-tych rokov boli vyvinuté mnohé ďalšie DNA algoritmy na riešenie rôznych NP-ťažkých problémov a pre malé prípady problémov boli aj v laboratóriu zrealizované. Aktuálny výskum sa venuje zvyšovaniu spoľahlivosti a rýchlosti uskutočňovania biochemických operácií a tiež vývoju šikovnejších algoritmických konceptov pre DNA algoritmy, ako je úplné prehľadávanie všetkých možností.

8.4 Silné a slabé stránky DNA počítača a vízie za horizont

Silnou stránkou DNA technológie je veľká miera miniaturizácie, čím sa dosahuje masívny paralelizmus spracovania údajov. Aj keď priebeh niektorých biochemických operácií trvá celé hodiny či dokonca dni, je počet pri tom paralelne uskutočnených operácií taký vysoký, že by simulácia na hociakom elektronickom počítači trvala roky.

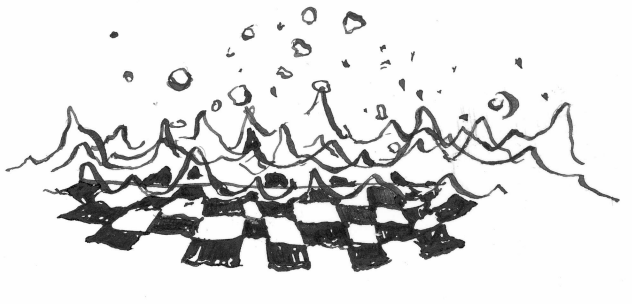
V súčasnosti DNA počítače ešte nekonkurujú elektronickým, pretože DNA technológia je len na počiatku. Uskutočnenie operácií trvá hodiny a dni a výsledky nie sú spoľahlivé. DNA sekvencie vieme čítať s 3% chybou, pri vykonávaní operácií dochádza k chybám, zlému nadviazaniu, medzerám pri nadviazaní atď. Slovom nevieme operácie vykonávať dostatočne

spoľahlivo a presne. Aby sme sa týmto problémom vyhli, využívame takzvanú redundanciu. Aby sa zvýšila pravdepodobnosť správneho vykonania operácií aspoň na jednom reťazci, musí sa zmnohonásobiť počet dvojitých reťazcov – nosičov informácií. Ale čím väčší počet operácií DNA počítač vykoná, tým je menšia pravdepodobnosť, že dostaneme správny výsledok. Vzhľadom na súčasný stav rozvoja DNA technológií to znamená, že pre mnohé prakticky zaujímavé prípady problémov nemáme k dispozícii dostatočný objem potrebného biologického materiálu, aby sme mohli spoľahlivo vykonať existujúce DNA algoritmy. Skeptici by mohli povedať: „Nemrhajme časom, a venujeme sa radšej niečomu užitočnému, kde vieme predvídať skoré uplatnenie.“ Moja odpoveď je, že to by bola najväčšia chyba, ktorú by mohol manažment vedy urobiť. Keby sme sa vo vede sústredili len na ciele, ktoré prinášajú celkom predvídateľný a dosiahnuteľný prospech, nikdy by sme nič podstatné neobjavili a pravdepodobne by sme sa ešte dnes šplhali po stromoch a nepoznali oheň. Pozrime sa na úroveň počítačovej techniky pred 40-50 rokmi. Počítač potreboval veľkú miestnosť a dennú údržbu, čo znemožňovalo dlhšie výpočty. Bojovali sme s chladením čoraz väčšmi sa zahrievajúcich zariadení a často sme zistili, že vo výsledkoch niečo nesedí, zavše sme museli celodenný výpočet opakovať. Programy sa zadávali do počítača takzvanými diernymi štítkami. Dierny štítok obsahoval jeden riadok programu (jeden príkaz) v binárnom kóde (diera alebo „nie diera“). Nešlo len o problém ako vyvinúť správny program. Keď mal program tisíce riadkov a niekto rozsypal krabicu s neočíslovanými diernymi štítkami, pozbieranie a utriedenie štítkov nemuselo byť najrýchlejším spôsob opätovného zostavenia programu. Koľkí verili v obrovský komerčný úspech počítača? Ale vývoj si nevšímal skeptikov. Kto dnes vie, čo všetko bude možné vďaka DNA technológiám? Na prelomenie štandardne používaného DES kryptosystému, ktorý sa považuje za bezpečný, potrebujú dnes DNA počítače 18 rokov [PRS05]. Ale keď sa zvýši spoľahlivosť a rýchlosť vykonávania biochemických operácií, je mysliteľné, aby to bolo len pár hodín. Možnosti na vylepšenie DNA technológií nesporne existujú. A napriek tomu, že krátkodobo nevidíme komerčné využitie, veda musí ísť cestou objavovania hraníc tejto technológie. Som presvedčený, že v tejto oblasti sa ešte dočkáme divov, ktoré nás potešia.

V súvislosti s biologickým spracovaním údajov formuloval Adleman aj celkom odlišnú predstavu, ktorej realizácia je ešte ďaleko za horizontom. Informácie sa nespracovávajú len v počítačoch alebo našich hlavách. Nachádzajú sa ako obyčajný prírodný fenomén v biologických alebo fyzikálnych systémoch. DNA molekula obsahuje okrem génov, ktoré slúžia

ako návody na vytváranie určitých bielkovín aj ďalšie informácie, ktoré riadia výber príslušného návodu. Nespráva sa v tomto ohľade jediná molekula ako samostatný počítač? Keď pochopíme lepšie jednotlivé programy a biologické mechanizmy, existuje možnosť programovať molekulu ako univerzálny počítač. Potom nebudeme potrebovať tony biomasy na uskutočnenie biologického spracovania informácií. Naše algoritmy budú vykonateľné na základe naprogramovania DNA molekuly.

Pre nebiológov, ktorí sa zaujímajú o základy manipulácie s DNA molekulou, doporučujeme učebnicu Böckenhauera a Bongartza [BB03]. Vynikajúci úvod do molekulárnej biológie pre začiatočníkov je učebnica Drlicu [Drl92]. Podrobný a s nadšením napísaný úvod do DNA počítania je v [PRS05]. Koncept je výstižne predstavený aj v [Hro04a].



Keď neočakávaš neočakávané,
nenájdeš nič drahocenné,
čo ťažko nájsť.

Heraklit

Kapitola 9

Kvantový počítač alebo počítanie v zázračnom svete mikročastíc

9.1 Prehistória a vytýčenie cieľa

Fyzika je zázračná veda. Keby som mal na gymnáziu dobrého učiteľa fyziky, pravdepodobne by som sa stal fyzikom. Ale nebanujem, že som sa stal informatikom. Keď preniknete dostatočne hlboko do základov vlastnej vednej disciplíny, nevyhnutne sa dotknete aj iných oblastí základného výskumu. Otvorí vám to prístup k spoločnému základu všetkých vedeckých disciplín a uvidíte všetko jasnejšie a vzrušujúcejšie ako keby ste sa na veci pozerali len úzkym pohľadom jednej vednej disciplíny[1]. Fyzika poskytuje širší a zároveň aj hlbší pohľad na svet. Nijaká iná vedecká disciplína, najmä v devätnástom a prvej polovici dvadsiateho storočia, neformovala náš pohľad na svet tak významne ako fyzika. Vzrušujúce

[1] Priveľká špecializácia a s tým spojený zúžený pohľad sú najväčším problémom dnešnej vedy, ktorá na jednej strane „chráni“ pred hlbokým pochopením a pred naozajstnými vedeckými úspechmi a na druhej strane vyvoláva vysokú netolerantnosť, ba až pohŕdanie medzi vedcami z rôznych vedeckých oblastí.

objavy, neočakávané zvraty a pozoruhodné výsledky boli na dennom poriadku fyzikálneho výskumu. Som presvedčený, že kvantová mechanika patrí k najväčším vedeckým objavom. Pochopiť a akceptovať ju bolo pre ľudí rovnako ťažké, ako v stredoveku sa vzdať predstavy, že zem je stredobodom vesmíru. Prečo narážalo uznanie kvantovej mechaniky na podobné problémy so svojím uznaním, ako práce Galilea Galileiho? Pretože zákony kvantovej mechaniky sú pravidlá správania sa elementárnych častíc a tieto sa nedajú zosúladiť s našimi skúsenosťami z makrosveta. Pozrime sa na najdôležitejšie princípy kvantovej mechaniky, ktorá rozbila svet klasickej fyziky:

- V makrosvete sa objekt v každom momente[2] nachádza len na jednom mieste. Neplatí to pre častice. Napríklad elektrón sa môže nachádzať na viacerých miestach v tom istom čase.

- Princíp kauzality hovorí, že každá akcia má svoju jednoznačne určenú reakciu, jednoznačne určené dôsledky. To neplatí vo svete elementárnych častíc. V istých situáciach (podľa teórie kvantovej mechaniky) tu vládne náhoda. Dôsledky niektorých akcií sa nedajú jednoznačne predpovedať. Existujú viaceré možnosti ako sa situácia bude vyvíjať a uskutoční sa náhodne jedna z týchto možností. Nemáme nijakú možnosť vypočítať a na základe toho predpovedať, ktorá z možností nastane. Vieme vypočítať len pravdepodobnosti, s ktorými nastanú jednotlivé možnosti. Z tohto dôvodu fyzici v týchto prípadoch hovoria o skutočne náhodných udalostiach.

- Princíp lokálnosti hovorí, že účinok akcie je vždy lokálny. V kvantovom svete môžu mať dve častice takú silnú vzájomnú väzbu, že nezávisle od ich vzájomnej vzdialenosti (hoci aj miliardu svetelných rokov), zmena stavu jednej z nich súčasne spôsobí zmenu stavu druhej z nich.

- Klasická fyzika hovorí, že keď je nejaká udalosť možná (teda keď udalosť má kladnú pravdepodobnosť výskytu), tak sa s príslušnou frekvenciou bude vyskytovať. Vo svete častíc je to inak. Dve možné udalosti, ktoré sa môžu vyskytnúť s kladnou pravdepodobnosťou, sa môžu vzájomne celkom zrušiť, podobne ako dve vlny, takže ani jedna z nich nenastane.

Ako mohli fyzici objaviť vôbec takéto „podivuhodné“ zákony, ba čo viac presvedčiť sa o ich vierohodnosti? V princípe rovnako, ako to robili aj vždy predtým. Vymysleli ich výskumníci s geniálnou intuíciou na základe

[2]Podľa teórie relativity je aj čas subjektívny (relatívny) pojem.

experimentov, pozorovaní a úvah. Svoje myšlienky vyjadrili v jazyku matematiky. Na základe matematického modelu boli fyzici schopní vytvoriť hypotézy. Keď sa všetky hypotézy potvrdili aj experimentálne, bol dobrý dôvod pokladať model za vierohodný. Experimentálne potvrdenie teórie kvantovej mechaniky trvalo mnoho rokov, pretože na potvrdenie niektorých hypotéz tejto teórie bol potrebný podstatný rozvoj v oblasti experimentálnej fyziky. Navyše tento rozvoj bol často finančne veľmi náročný.

Čo chceme povedať? Na priblíženie kvantovej mechaniky by nestačilo ani niekoľko kníh ako je táto. Potrebná matematika nie je ľahká a náročnosť sa ešte zvýši, keď sa usilujeme navrhnúť kvantové algoritmy na riešenie algoritmických úloh. Z týchto dôvodov sa uspokojíme s azda nie celkom ostrým obrazom hlavných konceptov správania sa častíc a toho, ako ich správanie využiť pri algoritmickom spracovaní informácií.

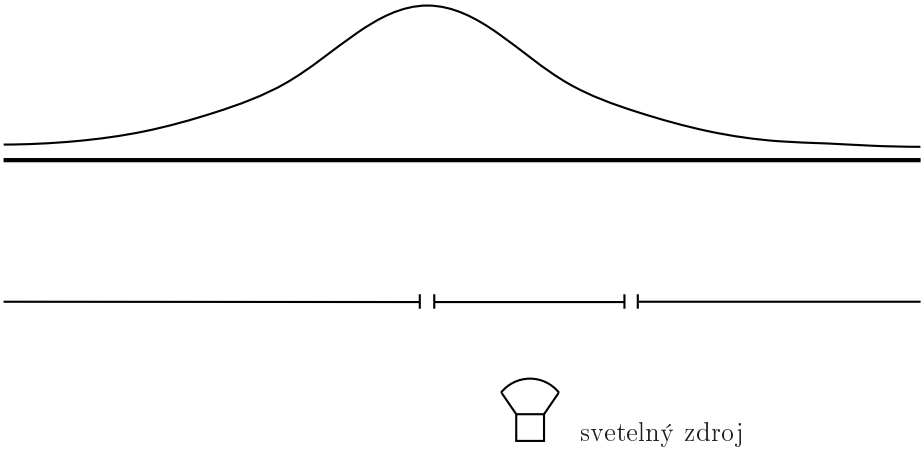
V nasledujúcej časti navštívime čarovný svet kvantovej mechaniky, všimneme si správanie sa častíc v experimentoch a pokúsime sa ho objasniť. V časti 9.3 vysvetlíme, ako sa dajú uložiť bity v kvantovom systéme a ako sa s nimi dá počítať. Budeme sa venovať aj problémom konštrukcie kvantového počítača. V odseku 9.4 uzavrieme kapitolu diskusiou o perspektívach kvantového počítania pri riešení ťažkých problémov a pri vývoji bezpečných kryptosystémov.

9.2 Krátka prechádzka čarovným svetom kvantovej mechaniky

Alica bola v krajine zázrakov len vo sne. Fyzici to majú oveľa ťažšie. Pri svojej práci sú denne konfrontovaní so zázračnou krajinou častíc. Nemôžu z nej utiecť tým, že sa prebudia. Musia sa zo všetkých síl snažiť vysvetliť aspoň sčasti prečudesné správanie sa častíc, a čo je najhoršie, musia pritom zabudnúť všetky svoje doterajšie predstavy o tom, ako funguje tento svet. Opustiť vlastné predstavy nie je ľahké a vyžaduje si to určitý čas a ešte ťažšie je presvedčiť iných, že staré predstavy nie sú celkom správne, že ich treba nahradiť neuveriteľnými a doterajším skúsenostiam odporujúcimi konceptmi. A fyzici ešte môžu byť šťastí, že uveriteľnosť fyzikálnych teorií sa nepotvrdzuje ľudovým hlasovaním. Ako postupovať, aby sme pre ľudí žijúcich v makrosvete predstavu sveta častíc, ktorá sa prieči zdravému rozumu, spravili aspoň trocha presvedčivú a akceptovateľnú? Začneme[3] s experimentmi, ktoré sa budeme usilovať vysvetliť.

[3] Ako to urobili kedysi fyzici.

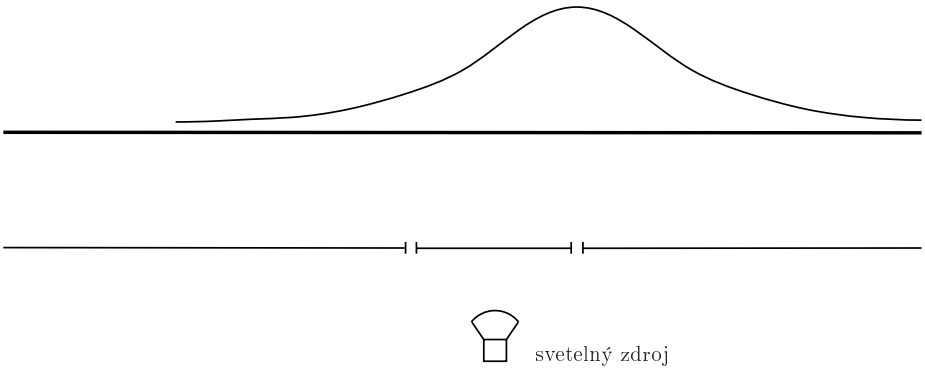
Skúsme najprv takzvaný experiment s dvoma štrbinami (obrázky 9.1, 9.2 a 9.3). Majme zdroj fotónov, z ktorého ich môžeme vysielať všetkými smermi. Oproti zdroju svetla bude stena s dvomi štrbinami, ktoré sa dajú zatvárať ako okno. V určitej vzdialenosti za stenou je fólia (hrubá čiara na obr.9.1), na ktorej vieme registrovať na ňu dopadajúce častice. Na obr. 9.1 je zobrazená situácia, keď je otvorená ľavá a zatvorená pravá štrbina. Krivka na obr. 9.1 znázorňuje, ako často sme na fólii zaregistrovali dopad častíc. Všetko je podľa našich predstáv. Podľa očakávania najčastejšie častice narážali na fóliu priamo oproti otvorenej štrbine. Krivka tam dosahuje najväčšiu hodnotu. Čím viac sa doprava alebo doľava vzďalujeme od štrbiny, tým menej častíc dopadá na fóliu, krivka na obe strany klesá. Podobne očakávaný výsledok experimentu dostaneme, keď je zatvorená ľavá štrbina a otvorená pravá (obr. 9.2). Výskyt častíc sa zvyšuje, čím sme bližšie k miestu, ktoré je presne oproti otvorenej štrbine. Podobne ako na obr. 9.1, krivka dosahuje najvyššiu hodnotu oproti otvoru a klesá na obe strany.



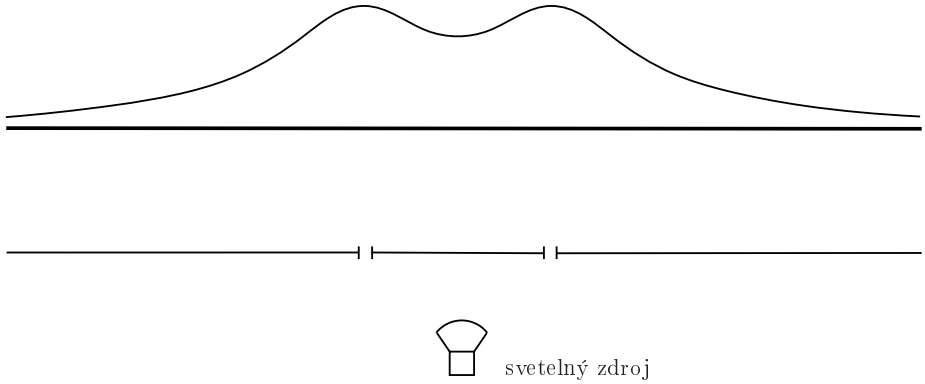
obr. 9.1

Očakávali by sme ale, že keď obe štrbiny otvoríme, bude výsledná početnosť výskytu častíc súčtom početností z obr. 9.1 a obr. 9.2, keď bola otvorená vždy len jedna štrbina. Tomu zodpovedajúca krivka početností je znázornená na obr. 9.3. Prekvapení zisťujeme, že to tak nie je. Na obr. 9.4 je znázornený pozorovaný výsledok experimentu, keď sú obe štrbiny otvorené.

Krivka početnosti výskytu z obr. 9.4 sa podstatne odlišuje od očakávanej krivky na obr. 9.3. A predsa, krivka z obr. 9.4 nie je fyzikovi neznáma alebo dokonca chaotická. Fyzik ihneď rozpozná, že zodpovedá interferen-



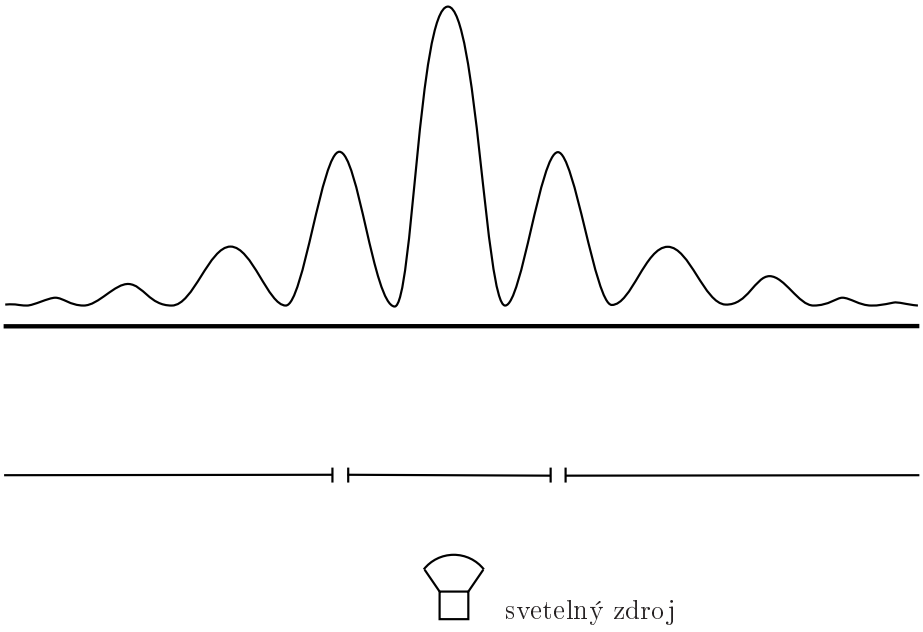
obr. 9.2



obr. 9.3

cii vĺn. Keby sme z každej štrbiny súčasne vypustili vlnu, na niektorých miestach sa vlny navzájom zrušia a na iných zasa zosilnia. Vravíme, že interferujú. Výsledok zosilnenia a zoslabenia zodpovedá presne krivke početnosti výskytu z obr. 9.4. Ako to vysvetliť? Fotóny (alebo iné častice) vypúšťame zo zdroja postupne, takže dva rôzne fotóny nemôžu vzájomne interferovať. Jediné vysvetlenie je, že každý fotón prejde súčasne oboma štrbinami a interferuje sám so sebou. Podstata tohto experimentu zodpovedá presne základom kvantového počítania. Častica je sčasti v ľavej a sčasti v pravej štrbine. Keď výskyt častice v ľavej štrbine reprezentuje hodnotu bitu 0 a výskyt v pravej štrbine reprezentuje hodnotu bitu 1, je hodnota bitu čiastočne 0 a čiastočne 1, čo v našom makrosvete nie je možné. Podstata matematického modelu kvantovej mechaniky je založená presne na tom, že častica sa do istej miery môže súčasne vyskytovať na viacerých miestach a interferovať sama so sebou. Výsledky predpove-

dané na základe tohto modelu sa presne zhodujú s pozorovaniami v experimentoch (obr. 9.4).



obr. 9.4

A to ešte nie je koniec. Experimentátor sa môže rozhodnúť, že podrobne preskúma správanie sa častíc. Ďalším zdrojom svetla si zboku posvieti na štrbiny, aby zistil, cez ktorú z nich častica prejde. A znovu zázrak! S prekvapením zistí, že každý fotón sa ukáže práve v jednej štrbine, nikdy nie v oboch súčasne. Krivka početnosti dopadu častíc sa tiež zmenila a bude zodpovedať pôvodne očakávanej krivke z obr. 9.3.

To sú ale prefikané častice! Správajú sa ako svätuškári. Keď ich pozorujeme, robia presne to, čo sa od nich očakáva. Keď ale na ne nehľadíme, robia všetko možné, len nie to, čo sa od nich očakáva. Môžete ľubovoľne často zamieňať pozorovanie a nepozorovanie štrbiny a zodpovedajúce krivky početností výskytu sa budú príslušne podľa toho meniť, ako je to znázornené na obr. 9.3 a obr. 9.4. Keď stlmíme bočný zdroj svetla slúžiaci na pozorovanie štrbiny do tej miery, že osvetlí len zlomok častíc, bude krivka početností výskytu častíc zmiešaním kriviek z obr. 9.3 a obr. 9.4. Čím viac svetla, tým bude viac podobná krivke z obr. 9.3. Čím bude svetla menej, tým viac bude podobná krivke z obr. 9.4. Ako vysvetlíme takéto správanie?

Pomocou kvantovej mechaniky. Nasledujúca všeobecne platná skutočnosť je veľmi dôležitá.

> *Nevieme vykonať pozorovanie alebo meranie bez toho, aby sme pritom neovplyvnili stav pozorovaného objektu a tým aj hodnoty, ktoré meriame*[4].

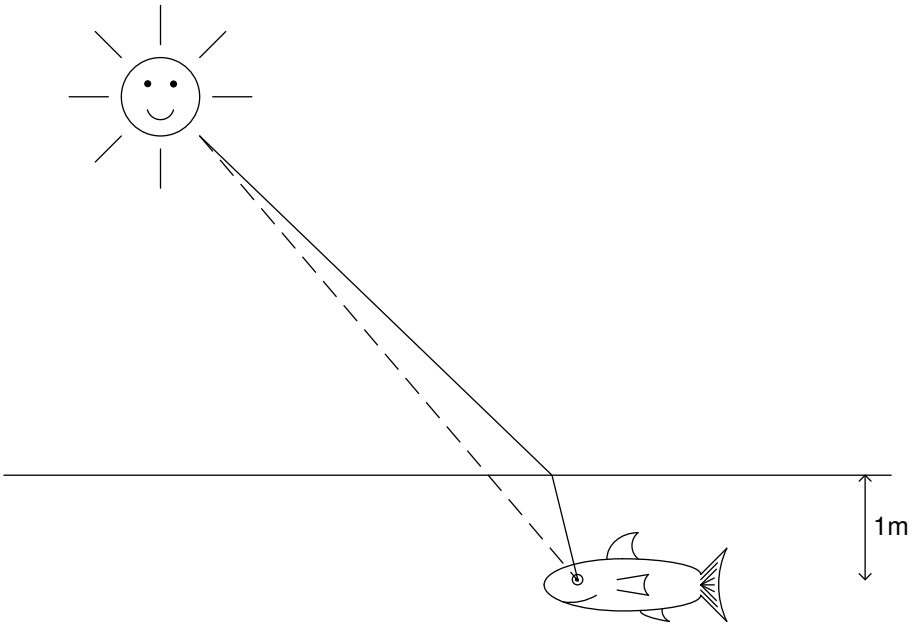
Práve to sa stalo v našom experimente. Bočný zdroj svetla ovplyvnil meranie tak, že každú pozorovanú časticu prechádzajúcu súčasne oboma štrbinami náhodne zafixoval do jednej z nich. Každé pozorovanie kvantovomechanického systému spôsobí, že vytvoríme takzvaný fixovaný **klasický stav**. Klasický stav zodpovedá tradičnému klasickému svetu. Častica je buď tu, buď tam, ale nikdy nie je na dvoch miestach zároveň. Kde sa pri našom pozorovaní z boku častica zafixuje, je náhodná udalosť, ktorú vieme modelovať pomocou zákonov kvantovej mechaniky. Nemôžeme ovplyvniť a ani jednoznačne predpovedať, ktorou štrbinou bude častica prechádzať. Vieme vypočítať len pravdepodobnosť, s ktorou sa objaví v jednej alebo druhej štrbine. Aká podstatná je táto skutočnosť pri tvorbe kvantovomechanických algoritmov, uvidíme v nasledujúcej kapitole.

Ak vás zneistil predstavený experiment s dvomi štrbinami, nič si z toho nerobte. Sami fyzici potrebovali mnoho rokov na to, kým sa zžili s týmito predstavami o správaní sa častíc. Presnejšie povedané až nástup novej generácie vo fyzikálnom výskume znamenal akceptovanie a strávenie kvantovej mechaniky. Pretože dúfam, že doterajšie čítanie knihy vás už dostatočne zocelilo, dovolím si predstaviť ešte jeden fyzikálny experiment.

Nasledujúci experiment zvýrazní významnú úlohu interferencie. Predstavme si prázdninovú idylku. Slnko svieti, na oblohe ani mráčka, bezvetrie a pred nami modré more, v ktorom 1 meter pod hladinou odpočíva neškodná ryba. Ruší ju iba do oka dopadajúci slnečný lúč. Aká je jeho dráha? Podľa známych fyzikálnych zákonov pôjde slnečný lúč zo slnka do oka po časovo najkratšej dráhe. Časovo najkratšia dráha neznamená priamku spájajúcu slnko a oko ryby (prerušovaná čiara na obr. 9.5). Pretože sa svetlo šíri vo vzduchu rýchlejšie ako vo vode, pôjde lúč radšej dlhšie vzduchom, aby si skrátil cestu vodou (plná čiara na obr. 9.5). Očividne lúč pri vstupe do vody mení smer.

Tento uhol vieme vypočítať. Zisťujeme, že slnečný lúč sa láme na hladine pod určitým uhlom. Uvažujme o rovnakom experimente s rybou odpočí-

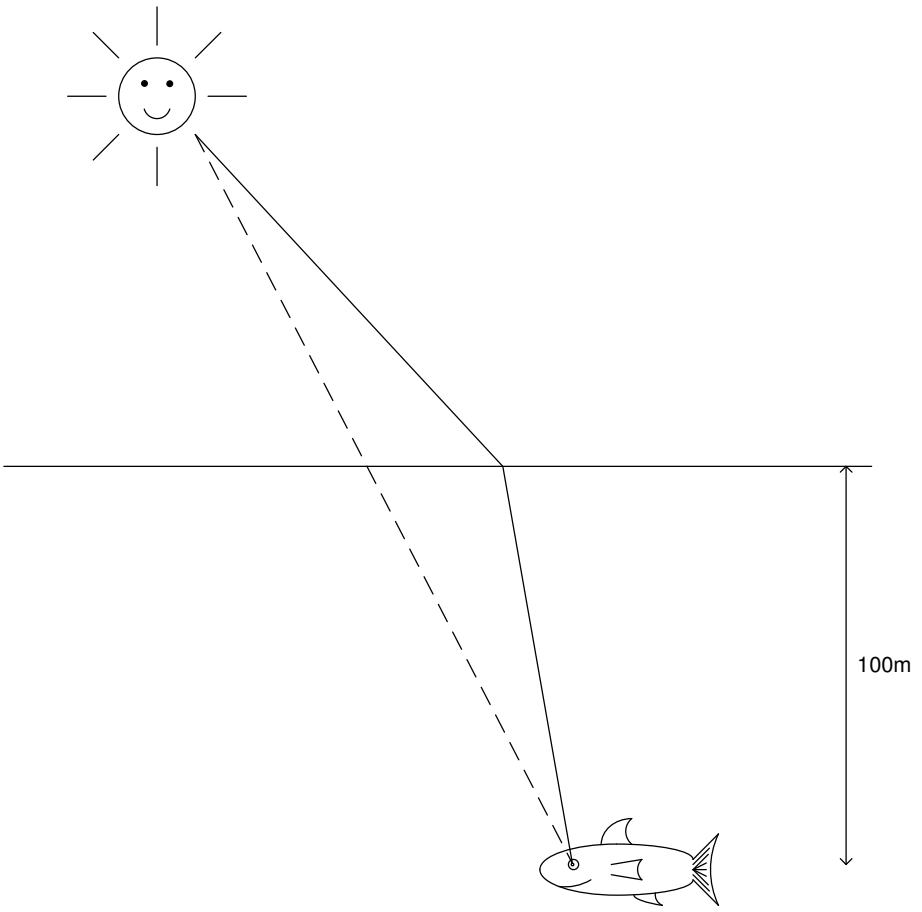
[4] Ak predpokladáme, že namerané hodnoty zodpovedajú presne realite, ide o veľmi idealizovaný model experimentu.



obr. 9.5

vajúcou 100 m pod hladinou. V tomto prípade by sa mala dráha vo vode ešte viac skrátiť. Preto sa posunie časovo najkratšia dráha lúča ešte viac doprava a bude sa pri vstupe do vody lámať pod väčším uhlom (obr. 9.6).

Aj keď sme veľkosť uhla na obr. 9.6 trocha zveličili, z nášho experimentu je nad slnko jasnejšie, že uhly, pod ktorými lúče vchádzajú do vody, sú rôzne. To predsa nie je s kostolným poriadkom! Ako môže vedieť lúč na hladine, či padne do oka rybe nachádzajúcej sa 1 m alebo 100 m pod hladinou a podľa toho zmeniť svoj smer? Alebo ešte lepšia otázka. Ako sa už pri štarte zo slnka môže rozhodnúť, komu padne do oka a na základe toho sa vydať časovo najkratšou trasou? S takýmito múdrymi časticami nik nepočítal. V klasickej fyzike síce môžeme pozorovať, že sa svetlo vždy šíri po časovo najkratšej trase, ale nevieme vysvetliť, ako to svetlo robí. Naproti tomu kvantová mechanika to vysvetliť vie. Slnečné lúče si prirodzene nemôžu vypočítať a naplánovať svoju trasu. Jednoducho sa pohybujú (vyžarujú) všetkými smermi a usilujú sa dosiahnuť oko ryby všetkými možnými trasami. Ibaže tie, ktoré neletia najkratšou trasou, vzájomne interferujú ako vlny, a tým sa navzájom zrušia tak, že ostane len jeden fotón na najkratšej ceste. Tento jediný fotón nakoniec dopadne rybe do oka. Aké je z toho poučenie? Predsa prebieha výpočet. Ale nepočíta slnečný lúč. Je to globálny výpočet podľa prírodných zákonov kvantovej



obr. 9.6

mechaniky, ktoré sú neustále prítomné v prírode a o ktorých pôsobení sa môžeme presvedčiť z pozorovania. Témou nasledujúcej časti je, ako využiť kvantovomechanické počítanie na strojové algoritmické počítanie.

9.3 Ako počítame vo svete častíc?

V kapitole 2 sme do určitej miery predstavili štruktúru počítača. Veľmi zhruba potrebujeme pamäť na ukladanie údajov a možnosť spracovávať (meniť) ich určitými operáciami. V klasickom počítači sme si pamätali postupnosti bitov. Prostredníctvom logických obvodov sme s postupnosťami bitov vedeli vykonávať aritmetické a textové operácie. Pri varení

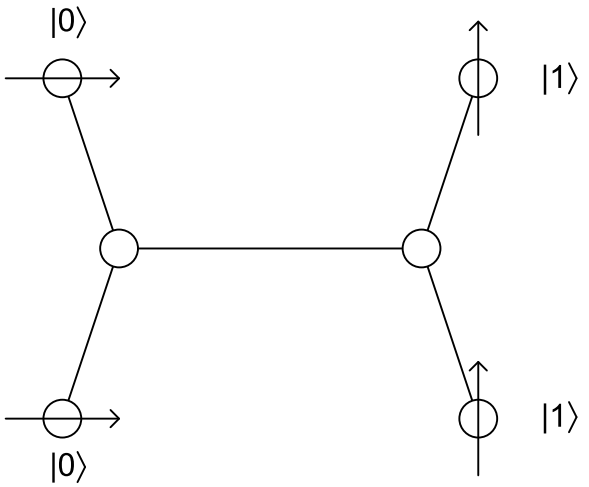
boli pamäťou nádoby všetkého druhu a hardvér na vykonávanie operácií boli rôzne kuchynské prístroje, ako šporák, rúra, mikrovlnka, mixér, atď. V biopočítači sme si pamätali údaje ako DNA sekvencie v reagenčných miskách a operácie s nimi sme vykonávali chemickými reakciami. Ako vytvoríme kvantový počítač? Už je jasné, že musíme určiť, akým spôsobom si budeme pamätať údaje a ako budeme s nimi vykonávať operácie.

Rovnako ako v klasickom počítači použijeme na reprezentáciu údajov bity. Aj tu budeme pracovať s registrami, ktoré nazveme **kvantové registre**. V kvantovom registri sú zapamätané **kvantové bity**. Aby sme ich odlíšili od klasických bitov 0 a 1 budeme ich označovať

$$|0\rangle \text{ a } |1\rangle.$$

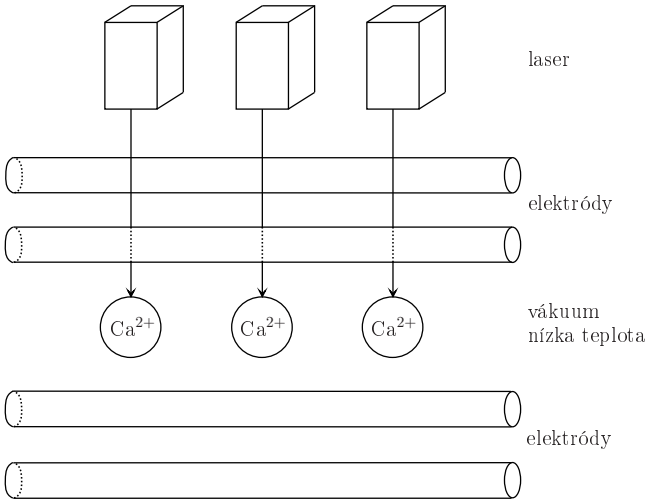
Je viacero možností, ako fyzikálne realizovať kvantové bity. Nasledujúce tri krátke odseky sú venované len čitateľom, pre ktorých je fyzika záľubou. Zvyšok kapitoly je možné čítať, aj keď ich preskočíte.

Prvá možnosť vytvorenia kvantového bitu je založená na využití *jadrovej magnetickej rezonancie*. Na obr. 9.7 je znázornený príklad použitia štyroch zo šiestich atómov v molekule ako kvantového registra. Keď sa molekula nachádza v magnetickom poli, smeruje spin atómov paralelne s magnetickým poľom. Smer paralelne s magnetickým poľom interpretujeme ako $|0\rangle$. Smer kolmo na magnetické pole interpretujeme ako $|1\rangle$. Použitím oscilujúcich magnetických polí vieme na týchto kvantových bitoch vykonávať operácie.



obr. 9.7

Ďalšia možnosť je *iónová pasca.* Ióny sú elektricky nabité molekuly alebo atómy (na obr. 9.8 sú nabité kladne, každému chýbajú práve dva elektróny). Pri teplote blízkej absolútnej nule sa ióny držia vo vákuu prostredníctvom elektromagnetického poľa v iónových pasciach.



obr. 9.8

Hodnotu $|0\rangle$ priradíme základnému stavu iónu a excitovanému stavu iónu priradíme hodnotu $|1\rangle$. Na takto vytvorených kvantových bitoch robíme operácie prostredníctvom lasera.

Ako počíta kvantový počítač a aké sú jeho výhody? V klasickom počítači sme mali bitový register, ktorý mohol mať hodnotu buď 0 alebo 1. V kvantovom registri je to inak. Súčasne môže mať obe hodnoty alebo len určitú časť z každej. Presne ako častica v experimente s dvomi štrbinami, ktorá čiastočne prechádzala aj cez pravú a čiastočne aj cez ľavú štrbinu. Ako to zapíšeme?

Hovoríme, že *kvantový bit* (obsah kvantového registra) je **superpozíciou** (alebo **kombináciou**) dvoch klasických bitov $|0\rangle$ a $|1\rangle$. Superpozíciu zapisujeme:

$$\alpha \cdot |0\rangle \quad + \quad \beta \cdot |1\rangle,$$

kde α a β sú komplexné čísla, pre ktoré platí:

$$|\alpha|^2 \leq 1, |\beta|^2 \leq 1 \text{ a } |\alpha|^2 + |\beta|^2 = 1.$$

$|\alpha|$ označuje absolútnu hodnotu α. Ak neviete, čo sú to komplexné čísla, nevzdávajte sa a bez problémov môžete pokračovať v čítaní, pretože v ďalšom používame len reálne α a β. Pre každé reálne číslo α, $|\alpha|^2 = \alpha^2$, takže

naše podmienky pre α a β sa zjednodušia na:

$$\alpha^2 \leq 1, \beta^2 \leq 1 \text{ a } \alpha^2 + \beta^2 = 1.$$

Hodnoty α a β sa nazývajú **amplitúdy** a vyjadrujú, do akej miery je hodnota kvantového bitu $|0\rangle$ alebo $|1\rangle$.

Presnejšia interpretácia je takáto:

> α^2 je pravdepodobnosť, že pri meraní bude obsahom kvantového registra $|0\rangle$.
> β^2 je pravdepodobnosť, že pri meraní bude obsahom kvantového registra $|1\rangle$.

Požiadavka $\alpha^2 + \beta^2 = 1$ je dôsledkom tejto interpretácie, pretože pre klasické stavy nie je iná možnosť ako $|0\rangle$ a $|1\rangle$.

Hoci s istotou vieme, že kvantový register sa nachádza v stave (superpozícii)

$$\alpha \cdot |0\rangle + \beta \cdot |1\rangle,$$

nemáme možnosť zistiť o tejto superpozícii úplne všetko. Inak povedané, hodnoty α a β sa nedajú zmerať. Keď vykonáme meranie kvantového bitu, dostaneme len klasické hodnoty $|0\rangle$ alebo $|1\rangle$. Meranie definitívne zničí pôvodnú superpozíciu. Je to presne rovnaké ako pri elektróne, ktorý letí súčasne cez dve rôzne štrbiny, ale keď ho pozorujeme, letí vždy len cez jednu.

V našej interpretácii je α^2 pravdepodobnosť, s ktorou pri meraní dostaneme $|0\rangle$ a β^2 je pravdepodobnosť, že výsledkom merania bude $|1\rangle$.

Príklad 9.1 Superpozícia

$$\frac{1}{\sqrt{2}} \cdot |0\rangle + \frac{1}{\sqrt{2}} \cdot |1\rangle$$

vyjadruje skutočnosť, že kvantový bit je s rovnakou pravdepodobnosťou v klasickom stave $|0\rangle$ či $|1\rangle$, lebo

$$\alpha^2 = \left(\frac{1}{\sqrt{2}}\right)^2 = \frac{1}{(\sqrt{2})^2} = \frac{1}{2} \quad \text{a} \quad \beta^2 = \left(\frac{1}{\sqrt{2}}\right)^2 = \frac{1}{2}.$$

Pri viacnásobnom opakovaní merania superpozície preto nameriame hodnoty $|0\rangle$ a $|1\rangle$ rovnako často. □

Príklad 9.2 Superpozícia

$$\frac{1}{\sqrt{3}} \cdot |0\rangle + \sqrt{\frac{2}{3}} \cdot |1\rangle$$

vyjadruje, že pri meraní dostaneme výsledok $|0\rangle$ s pravdepodobnosťou

$$\alpha^2 = \left(\frac{1}{\sqrt{3}}\right)^2 = \frac{1}{3}$$

a výsledok $|1\rangle$ s pravdepodobnosťou

$$\beta^2 = \left(\sqrt{\frac{2}{3}}\right)^2 = \frac{2}{3}.$$

□

Úloha 9.1 Navrhnite superpozíciu kvantového bitu, ktorej meraním dostaneme hodnotu $|1\rangle$ s pravdepodobnosťou $\frac{1}{4}$ a hodnotu $|0\rangle$ s pravdepodobnosťou $\frac{3}{4}$.

Ako je to vo všeobecnom prípade? Keď máme n kvantových registrov, máme 2^n možných klasických obsahov. Superpozícia n kvantových bitov znamená, že register je súčasne vo všetkých 2^n klasických stavoch, pričom v každom z nich je s určitou pravdepodobnosťou. Jediná podmienka je, že súčet týchto 2^n pravdepodobností musí byť 1.

Príklad 9.3 Analyzujme dva kvantové registre. Všetky ich možné obsahy sú tieto:

$$00, \quad 01, \quad 10 \quad \text{a} \quad 11.$$

V prípade dvoch kvantových registrov je obsah pamäte kvantového počítača superpozícia

$$\alpha \cdot |00\rangle + \beta \cdot |01\rangle + \gamma \cdot |10\rangle + \delta \cdot |11\rangle,$$

kde

$$\alpha^2 \leq 1,\ \beta^2 \leq 1,\ \gamma^2 \leq 1,\ \delta^2 \leq 1 \ \text{ a } \ \alpha^2 + \beta^2 + \gamma^2 + \delta^2 = 1.$$

Konkrétna superpozícia

$$\frac{1}{2} \cdot |00\rangle + \frac{1}{2} \cdot |01\rangle + \frac{1}{2} \cdot |10\rangle + \frac{1}{2} \cdot |11\rangle$$

s $\alpha = \beta = \gamma = \delta = \frac{1}{2}$ opisuje situáciu, v ktorej každý z možných obsahov nameriame s rovnakou pravdepodobnosťou

$$\alpha^2 = \beta^2 = \gamma^2 = \delta^2 = \left(\frac{1}{2}\right)^2 = \frac{1}{4}.$$

Analyzujme superpozíciu

$$0 \cdot |00\rangle + 0 \cdot |01\rangle + 0 \cdot |10\rangle + 1 \cdot |11\rangle.$$

Pretože $\alpha = \beta = \gamma = 0$ a $\delta^2 = 1^2 = 1$, každé meranie dá jednoznačne výsledok

$$|11\rangle.$$

Pri superpozícii

$$\frac{1}{\sqrt{2}} \cdot |00\rangle + 0 \cdot |01\rangle + 0 \cdot |10\rangle + \frac{1}{\sqrt{2}} \cdot |11\rangle$$

nameriame len dva výsledky $|00\rangle$ a $|11\rangle$, oba s rovnakou pravdepodobnosťou $\frac{1}{2}$. □

Úloha 9.2 Ako vyzerá vo všeobecnosti superpozícia troch kvantových bitov?

a) Napíšte superpozíciu troch kvantových bitov, v ktorej meraním s rovnakou pravdepodobnosťou dostaneme ľubovoľný klasický obsah troch bitových registrov.
b) Nájdite superpozíciu troch kvantových bitov, pre ktorú meraním určíme klasický obsah $|111\rangle$ s pravdepodobnosťou $\frac{1}{2}$, hodnotu $|000\rangle$ s pravdepodobnosťou $\frac{1}{4}$ a každú zo zvyšných hodnôt s rovnakou pravdepodobnosťou.

Aké operácie sú možné v kvantovom svete? Ako sa v jednom kroku kvantového výpočtu dá zmeniť superpozícia na inú superpozíciu? Možné sú také výpočtové kroky, ktoré dostaneme ako vynásobenie superpozície (reprezentovanej vektorom) určitými maticami. Tieto matice majú špeciálnu vlastnosť, že zo superpozície vytvoria novú superpozíciu.

Zvyšok tejto časti je určený len čitateľom so záujmom o matematiku. Ostatní môžu preskočiť na časť 9.4.

Superpozíciu

$$\alpha_1 \cdot |00\rangle + \alpha_2 \cdot |01\rangle + \alpha_3 \cdot |10\rangle + \alpha_4 \cdot |11\rangle$$

si môžeme predstaviť ako stĺpcový vektor

$$\begin{pmatrix} \alpha_1 \\ \alpha_2 \\ \alpha_3 \\ \alpha_4 \end{pmatrix}.$$

Keď chceme zapísať stĺpcový vektor v riadku, urobíme to takto: $(\alpha_1, \alpha_2, \alpha_3, \alpha_4)^T$, kde T označuje transpozíciu. Riadkový vektor sa dá vynásobiť stĺpcovým nasledujúco:

$$(\beta_1, \beta_2, \beta_3, \beta_4) \cdot \begin{pmatrix} \alpha_1 \\ \alpha_2 \\ \alpha_3 \\ \alpha_4 \end{pmatrix} = \alpha_1\beta_1 + \alpha_2\beta_2 + \alpha_3\beta_3 + \alpha_4\beta_4.$$

Výsledkom je komplexné číslo (nie vektor). Maticu $n \times n$ si môžeme predstaviť ako n riadkových vektorov. Napríklad matica 4×4

$$M = \begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{pmatrix}$$

sa skladá zo štyroch riadkových vektorov

$$\begin{gathered} (a_{11}, a_{12}, a_{13}, a_{14}) \\ (a_{21}, a_{22}, a_{23}, a_{24}) \\ (a_{31}, a_{32}, a_{33}, a_{34}) \\ (a_{41}, a_{42}, a_{43}, a_{44}). \end{gathered}$$

Takže vynásobením matice M stĺpcovým vektorom $\alpha = (\alpha_1, \alpha_2, \alpha_3, \alpha_4)^T$ dostaneme zasa stĺpcový vektor $\mu = (\mu_1, \mu_2, \mu_3, \mu_4)^T$, v ktorom je i-ta pozícia súčinom i-teho riadkového vektora v M s α. Presnejšie:

$$M \cdot \begin{pmatrix} \alpha_1 \\ \alpha_2 \\ \alpha_3 \\ \alpha_4 \end{pmatrix} = \begin{pmatrix} \mu_1 \\ \mu_2 \\ \mu_3 \\ \mu_4 \end{pmatrix},$$

kde

$$\begin{aligned} \mu_i &= (a_{i,1}, a_{i,2}, a_{i,3}, a_{i,4}) \cdot \begin{pmatrix} \alpha_1 \\ \alpha_2 \\ \alpha_3 \\ \alpha_4 \end{pmatrix} \\ &= a_{i,1} \cdot \alpha_1 + a_{i,2} \cdot \alpha_2 + a_{i,3} \cdot \alpha_3 + a_{i,4} \cdot \alpha_4. \end{aligned}$$

Aplikovanie M na superpozíciu chápeme ako jeden výpočtový krok. Ak je pre každú superpozíciu α (reprezentovanú stĺpcovým vektorom) výsledok $M \cdot \alpha$ tiež superpozíciou (v našom príklade to znamená, že $\mu_1^2 + \mu_2^2 + \mu_3^2 + \mu_4^2 = 1$), potom je M povolená operácia.

V nasledujúcom príklade ukážeme, ako sa dajú na kvantovom počítači generovať náhodné bity.

Príklad 9.4 Máme k dispozícii jeden kvantový register. Začneme s „klasickou“ superpozíciou

$$|0\rangle = 1 \cdot |0\rangle + 0 \cdot |1\rangle.$$

Vykonáme jeden výpočtový krok: vynásobíme túto superpozíciu *Hadamardovou maticou*

$$H_2 = \begin{pmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{pmatrix}.$$

Výsledkom násobenia bude:

$$\begin{pmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{pmatrix} \cdot \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 1 \cdot \frac{1}{\sqrt{2}} + 0 \cdot \frac{1}{\sqrt{2}} \\ 1 \cdot \frac{1}{\sqrt{2}} + 0 \cdot \left(-\frac{1}{\sqrt{2}}\right) \end{pmatrix} = \begin{pmatrix} \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{pmatrix}.$$

Dostali sme superpozíciu

$$\frac{1}{\sqrt{2}} \cdot |0\rangle + \frac{1}{\sqrt{2}} \cdot |1\rangle.$$

Ak meriame túto superpozíciu, s rovnakou pravdepodobnosťou $\frac{1}{2}$ dostaneme klasický bit $|0\rangle$ alebo $|1\rangle$.

V prípade, že začneme s „klasickou“ superpozíciou

$$|1\rangle = 0 \cdot |0\rangle + 1 \cdot |1\rangle,$$

ktorú vynásobíme opäť maticou H_2, dostaneme

$$\begin{pmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{pmatrix} \cdot \begin{pmatrix} 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \cdot \frac{1}{\sqrt{2}} + 1 \cdot \frac{1}{\sqrt{2}} \\ 0 \cdot \frac{1}{\sqrt{2}} + 1 \cdot \left(-\frac{1}{\sqrt{2}}\right) \end{pmatrix} = \begin{pmatrix} \frac{1}{\sqrt{2}} \\ -\frac{1}{\sqrt{2}} \end{pmatrix}.$$

Výsledkom je superpozícia

$$\frac{1}{\sqrt{2}} \cdot |0\rangle - \frac{1}{\sqrt{2}} \cdot |1\rangle.$$

Pretože platí $\alpha^2 = \left(\frac{1}{\sqrt{2}}\right)^2 = \frac{1}{2}$ a $\beta^2 = \left(-\frac{1}{\sqrt{2}}\right)^2 = \frac{1}{2}$, výsledky merania $|0\rangle$ aj $|1\rangle$ dostaneme s rovnakou pravdepodobnosťou $\frac{1}{2}$. Čo to znamená? V oboch prípadoch sme dostali pri meraní náhodný bit, ale nie sme schopní rozlíšiť, ktorú z dvoch superpozícií $\frac{1}{\sqrt{2}} \cdot |0\rangle + \frac{1}{\sqrt{2}} \cdot |1\rangle$ a $\frac{1}{\sqrt{2}} \cdot |0\rangle - \frac{1}{\sqrt{2}} \cdot |1\rangle$ sme merali. □

Úloha 9.3 Nájdite ešte aspoň dve ďalšie superpozície kvantového bitu, v ktorých s rovnakou pravdepodobnosťou nameriate hodnoty $|0\rangle$ a $|1\rangle$.

Úloha 9.4 (tvrdý oriešok) Dokážte, že matica H_2 má tú vlastnosť, že pre každú superpozíciu $\alpha \cdot |0\rangle + \beta \cdot |1\rangle$ je

$$\begin{pmatrix} \gamma \\ \delta \end{pmatrix} := H_2 \cdot \begin{pmatrix} \alpha \\ \beta \end{pmatrix}$$

tiež superpozícia, že teda platí $\gamma^2 + \delta^2 = 1$.

Úloha 9.5 (tvrdý oriešok) Výpočet kvantového počítača je reverzibilný[5]. Keď nevykonávame merania, ktoré by zničili dosiahnuté superpozície, je možné použitím vhodných výpočtových krokov nechať výpočet prebiehať naspäť do počiatočného stavu. V príklade 9.4 to znamená, že existuje matica M rozmeru 2×2 taká, že

$$M \cdot \begin{pmatrix} \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \quad \text{a} \quad M \cdot \begin{pmatrix} \frac{1}{\sqrt{2}} \\ -\frac{1}{\sqrt{2}} \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \end{pmatrix}.$$

Nájdite maticu M.

Teraz aspoň do určitej miery chápeme, ako sa dá počítať vo svete častíc. Rôznym stavom častíc (napríklad spinu) musíme vhodným spôsobom priradiť hodnoty 0 a 1. Kvantové operácie vykonávame so superpozíciami kvantových bitov. Matematicky vykonávame operácie ako vynásobenie vektora, reprezentujúceho superpozíciu, maticou. Povolené sú len matice, ktoré transformujú superpozíciu zase na superpozíciu. V kvantovom svete sa dajú uskutočniť všetky operácie, založené na takýchto maticiach. Pritom je zrejmá jedna z najdôležitejších výhod kvantového počítača. Keď počítame s n kvantovými bitmi, máme superpozíciu všetkých 2^n možných klasických obsahov n kvantových bitov:

$$\alpha_0 \cdot |00\ldots0\rangle + \alpha_1 \cdot |00\ldots01\rangle + \ldots + \alpha_{2^n-1} \cdot |11\ldots1\rangle.$$

V jednom výpočtovom kroku sa zmení celá superpozícia. Simulovať krok výpočtu kvantového počítača na klasickom počítači nevieme lepšie ako vynásobením 2^n-rozmerného vektora $(\alpha_0, \alpha_1, \ldots, \alpha_{2^n-1})^T$ maticou rozmeru 2^n x 2^n. Pri takejto simulácii kroku kvantového výpočtu s n bitmi potrebujeme vzhľadom na n exponenciálne veľa aritmetických operácií.

Ďalšia výhoda kvantového počítania je už spomínaná interferencia, ktorá použitím vhodných kvantových operácií umožňuje vzájomné zrušenie niektorých možností a súčasné zvýšenie pravdepodobnosti výskytu iných.

Dnes poznáme viaceré problémy, pri ktorých sú kvantové algoritmy podstatne efektívnejšie ako najlepšie známe klasické algoritmy. Žiaľ matematické a algoritmické vedomosti potrebné na ich vysvetlenie sú príliš zložité na prezentovanie v zrozumiteľnej forme bez použitia matematického formalizmu. Preto len napríklad spomenieme, že existuje efektívny kvantový algoritmus na faktorizáciu[6] prirodzených čísiel. Ako už vieme, faktorizovať nie sme efektívne schopní pomocou klasických algoritmov. To predstavuje nebezpečenstvo pre kryptografiu s verejnými kľúčmi, ktorá je založená na výpočtovej obťažnosti faktorizácie. Ale netreba sa plašiť, môžeme ďalej pokojne spávať. Najväčšie vytvorené kvantové počítače majú najviac sedem kvantových bitov, teda dokážu pracovať len s číslami, ktoré vieme reprezentovať siedmimi bitmi. A ako vieme, v kryptografii

[5] Dá sa obrátiť.

[6] rozklad na prvočinitele

s verejným kľúčom sa pracuje s číslami, ktoré majú niekoľko tisíc bitov. Budúcnosť ukáže, či je kvantové počítanie užitočné pre reálne spracovanie údajov. To je téma nasledujúcej časti.

9.4 Čo prinesie budúcnosť?

Odpoveď nepozná nik. Aby sme si lepšie ozrejmili perspektívy kvantového počítania, objasníme, v čom spočívajú problémy pri vývoji tejto technológie. Matematický model kvantového sveta nám poskytol nástroj na vytvorenie kvantových algoritmov. Tieto algoritmy sú pri riešení niektorých problémov neočakávane efektívne. Pre výpočty s nimi potrebujeme kvantový počítač s takým počtom kvantových bitov, aká je veľkosť spracovávaných údajov. Máme už aj predstavu, ako stavom častíc realizovať kvantové bity. V čom teda väzí problém? Kvantový počítač je extrémne citlivý, vlastne je citlivejší ako hocičo, čo poznáme z klasického sveta. Povedať niekomu: „Si citlivejší ako kvantový počítač“, je hrubá urážka. Keď do počítajúceho kvantového počítača prenikne jediná častica (napríklad elektrón), nenávratne sa zmení doteraz prebiehajúci výpočet. Superpozíciu nevieme znovu zrekonštruovať a celý výpočet musíme spustiť odznova. Rovnako ako meranie ovplyvní meraný objekt, aj každá častica, ktorá prenikne do kvantového počítača, ovplyvní jeho výpočet. Kvantový počítač preto musíme celkom izolovať od okolitého prostredia. Ale to je ťažšia úloha ako vybudovať nepreniknuteľný trezor. Častice sú všade, aj v materiáli, ktorý by sme použili na izoláciu v reálnom svete. To je dôvod používania vákua, teplôt blízkych absolútnej nule a ďalších opatrení. Vieme, že nijaký systém nedokážeme izolovať od vplyvov prostredia ľubovoľne dlho. Úloha je izolovať ho aspoň na zlomok sekundy, pretože kvantový výpočet môže prebehnúť veľmi rýchlo. Základná predstava nie je skonštruovať všeobecný kvantový osobný počítač (kvantové-PC) použiteľné na rôzne aplikácie. Cieľom je postaviť jednoúčelový špecializovaný kvantový počítač, ktorý rieši len jednu špecifickú výpočtovú úlohu. To znamená, aby sme mohli vykonať jeden konkrétny kvantový algoritmus, musíme preň vytvoriť kvantový počítač, ktorý nebude okrem tejto konkrétnej úlohy riešiť nič iné. Fyzici sú schopní skonštruovať kvantové počítače s malým počtom bitov (3 až 7) a vykonať na nich niektoré kvantové algoritmy. Situácia je do určitej miery podobná DNA-počítaniu. V súčasnosti táto technológia v nijakom prípade nekonkuruje klasickým počítačom. V momente, keď objavíme lepšiu technológiu na vytvorenie kvantových počítačov sa situácia dramaticky zmení. Budeme musieť revidovať definíciu prakticky riešiteľného a neriešiteľného. Teoretici už majú

takúto novú teóriu zložitosti pripravenú vo svojich šuplíkoch.

Možnosť efektívne faktorizovať by zmenila do veľkej miery kryptografiu. Doterajšia kryptografia s verejnými kľúčmi by už nebola bezpečná voči protivníkovi, ktorý by na faktorizáciu použil kvantový počítač. Alebo by predsa bola? Možno sa podarí skonštruovať kvantový počítač len s niekoľkými stovkami bitov, takže pri použití čísiel s niekoľko tisíc desiatkovými ciframi budeme kryptografiu s verejným kľúčom ďalej úspešne používať. Je jedno, ako to dopadne, takýto pokrok nemôžeme chápať negatívne, aj keby sme kvôli nemu museli doterajšiu kryptografiu hodiť do koša. Každý pokrok zo sebou prináša aj nové možnosti. Kvantové efekty umožňujú realizovať kryptosystémy s veľmi vysokou mierou bezpečnosti. Ich princíp je založený na možnosti vytvoriť tzv. previazanú superpozíciu, ktorá má špeciálnu vlastnosť, že bez ohľadu na aktuálnu vzdialenosť oboch častíc, keď zmeriame stav jednej častice neskorším meraním stavu druhej častice, dostaneme rovnaký stav druhej častice ako sme namerali pri prvej častici. Takýmto spôsobom sa dve strany môžu dohodnúť na náhodnom bite, ktorý použijú ako kľúč. V praxi si to môžeme predstaviť takto: príjemca a odosielateľ majú každý jednu zo vzájomne previazaných častíc. Obaja vykonajú merania na svojej častici. Je jedno, ktorý z nich bude merať ako prvý. Dôležité je, že obaja namerajú ten istý klasický stav (klasický bit 0 alebo 1). Je tak, pretože prvé z meraní spôsobí, že obe častice zmenia svoj stav na rovnaký klasický stav. Pri druhom z meraní odmeriame už len tento klasický stav. Ak prijímateľ a odosielateľ uvedeným spôsobom vygenerujú náhodnú postupnosť bitov, môžu ju použiť ako kľúč v symetrickom kryptosystéme. Tento kvantový efekt predpovedali fyzici už pred mnohými rokmi. Albert Einstein neveril, že sa túto predpoveď podarí experimentálne overiť. Argumentoval, že tento jav, nazývaný teleportácia, protirečí princípu lokálnosti fyzikálnych účinkov. Cieľom experimentu je ukázať, že po zmeraní prvej častice stav druhej častice skolabuje do zodpovedajúceho klasického stavu rýchlejšie, ako prejde svetlo z miesta prvej častice na miesto druhej častice. Takýmto spôsobom sa chceme presvedčiť o nemožnosti ovplyvnenia stavu jednej častice tým, že by sa jej „poslala“ informácia o zmene stavu druhej častice, pretože rýchlosť svetla sa považuje za hornú hranicu rýchlosti. Experimentálne overenie bolo veľmi ťažké kvôli vysokej rýchlosti svetla. Napriek tomu vedci ho boli schopní úspešne uskutočniť na vzdialenosť 600 m ponad rieku Dunaj neďaleko Viedne.

Časy prvých objaviteľov sa neskončili a nikdy sa neskončia. Veda je vzrušujúca a môžeme sa tešiť ešte na mnoho dobrodružstiev a pravých zázrakov.

Návody na riešenie vybraných úloh

Úloha 9.1 Uvažujme superpozíciu:

$$\frac{1}{2}\sqrt{3}\cdot|0\rangle - \frac{1}{2}\cdot|1\rangle \ .$$

Je zrejmé, že

$$\left(\frac{1}{2}\sqrt{3}\right)^2 = \frac{1}{4}\cdot 3 = \frac{3}{4} \text{ a } \left(-\frac{1}{2}\right)^2 = \frac{1}{4} \ .$$

Takže klasickú hodnotu $|1\rangle$ nameriame s pravdepodobnosťou $\frac{1}{4}$. Vidíme, že superpozícia

$$\left(-\frac{\sqrt{3}}{2}\right)\cdot|0\rangle + \frac{1}{2}\cdot|1\rangle$$

tiež spĺňa naše požiadavky. Môžeme sa zamyslieť, ktoré ďalšie superpozície $\alpha\cdot|0\rangle + \beta\cdot|1\rangle$ spĺňajú podmienky $\alpha^2 = 3/4$ a $\beta^2 = 1/4$. Je očividné, že meraním nikdy nezistíme superpozíciu, v ktorej sa nachádza kvantový systém.

Úloha 9.2 Vo všeobecnosti máme pre tri bity $2^3 = 8$ možných rôznych obsahov:

$$000, 001, 010, 011, 100, 101, 110 \text{ a } 111.$$

Teda každý stav kvantového registra s tromi bitmi je superpozícia

$$\alpha_0\cdot|000\rangle + \alpha_1\cdot|001\rangle + \alpha_2\cdot|010\rangle + \alpha_3\cdot|011\rangle$$

$$+\alpha_4\cdot|100\rangle + \alpha_5\cdot|101\rangle + \alpha_6\cdot|110\rangle + \alpha_7\cdot|111\rangle$$

ôsmich klasických stavov, kde

$$\alpha_i^2 \leq 1 \text{ pre } i = 0, 1, \ldots, 7 \text{ a } \sum_{i=0}^{7}\alpha_i^2 = 1.$$

a) Keď je rovnaká pravdepodobnosť, že nameriame hociktorý z ôsmich klasických stavov, potom musí platiť $\alpha_0^2 = \alpha_1^2 = \alpha_2^2 = \ldots = \alpha_7^2 = \frac{1}{8}$. Pretože

$$\frac{1}{8} = \frac{2}{16} = \left(\sqrt{\frac{2}{16}}\right)^2 = \left(\frac{\sqrt{2}}{4}\right)^2 \ ,$$

dostaneme

$$\alpha_0 = \alpha_1 = \alpha_2 = \ldots = \alpha_7 = \frac{\sqrt{2}}{4} \ ,$$

ako jednu z možností. Viete navrhnúť ďalšie možnosti?

b) Okrem obsahov $|111\rangle$ a $|000\rangle$ existuje ešte šesť ďalších možných obsahov. Keď $|000\rangle$ pozorujeme s pravdepodobnosťou $\frac{1}{2}$ a $|111\rangle$, pozorujeme s pravdepodobnosťou $\frac{1}{4}$, je pravdepodobnosť pozorovania všetkých ostatných možností $\frac{1}{4}$ (súčet všetkých pravdepodobností musí byť 1). Pravdepodobnosť $\frac{1}{4}$ musí byť rovnomerne rozdelená medzi zvyšných šesť obsahov. Takže pravdepodobnosť pozorovania niektorého zo zvyšných obsahov je:

$$\frac{1/4}{6} = \frac{1}{24} = \frac{6}{144}.$$

Nasledujúca superpozícia spĺňa naše požiadavky na pravdepodobnosti pozorovania daných klasických stavov:

$$\frac{1}{2}\cdot|000\rangle + \frac{1}{12}\sqrt{6}\cdot|001\rangle + \frac{1}{12}\sqrt{6}\cdot|010\rangle + \frac{1}{12}\sqrt{6}\cdot|011\rangle +$$
$$\frac{1}{12}\sqrt{6}\cdot|100\rangle + \frac{1}{12}\sqrt{6}\cdot|101\rangle + \frac{1}{12}\sqrt{6}\cdot|110\rangle + \frac{1}{2}\sqrt{2}\cdot|111\rangle.$$

Úloha 9.4 Vynásobíme

$$\begin{aligned} H_2\cdot\begin{pmatrix}\alpha\\ \beta\end{pmatrix} &= \begin{pmatrix}\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}}\\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}}\end{pmatrix}\cdot\begin{pmatrix}\alpha\\ \beta\end{pmatrix}\\ &= \begin{pmatrix}\frac{1}{\sqrt{2}}\cdot\alpha + \frac{1}{\sqrt{2}}\cdot\beta\\ \frac{1}{\sqrt{2}}\cdot\alpha - \frac{1}{\sqrt{2}}\cdot\beta\end{pmatrix} = \begin{pmatrix}\gamma\\ \delta\end{pmatrix}\end{aligned}$$

Našou úlohou je ukázať, že

$$\gamma\cdot|0\rangle + \delta\cdot|1\rangle$$

je superpozícia, t.j., že platí $\gamma^2+\delta^2=1$.

$$\begin{aligned}\gamma^2+\delta^2 &= \left(\frac{1}{\sqrt{2}}\cdot\alpha + \frac{1}{\sqrt{2}}\cdot\beta\right)^2 + \left(\frac{1}{\sqrt{2}}\cdot\alpha - \frac{1}{\sqrt{2}}\cdot\beta\right)^2\\ &= \frac{\alpha^2}{2} + 2\cdot\frac{1}{\sqrt{2}}\cdot\frac{1}{\sqrt{2}}\cdot\alpha\cdot\beta + \frac{\beta^2}{2} + \frac{\alpha^2}{2} - 2\cdot\frac{1}{\sqrt{2}}\cdot\frac{1}{\sqrt{2}}\cdot\alpha\cdot\beta + \frac{\beta^2}{2}\\ &= \alpha^2+\beta^2.\end{aligned}$$

Pretože sme predpokladali, že $\alpha\cdot|0\rangle + \beta|1\rangle$ je superpozícia, platí $\alpha^2+\beta^2=1$. Preto platí aj $\gamma^2+\delta^2=1$ a vektor $(\gamma,\delta)^T$ reprezentuje superpozíciu $|0\rangle$ a $|1\rangle$.

Život sa dá pochopiť, len keď sa obzrieme nazad,
ale žiť ho môžeme len pozerajúc sa vpred.

Sören Kierkegaard

Kapitola 10

Ako sa správne rozhodovať pre neznámu budúcnosť alebo ako prekabátiť prefíkaného protivníka

10.1 Aké objavy nás tu očakávajú?

Doteraz riešené algoritmické úlohy sa vyznačovali tým, že sme mali pre daný prípad problému (konkrétnu úlohu) nájsť správne riešenie. To znamená, že sme mali k dispozícii od začiatku výpočtu všetky údaje o vstupe, ktoré sme potrebovali na vypočítanie riešenia. V praxi existuje ale množstvo úloh, pri riešení ktorých máme na začiatku k dispozícii len časť údajov. V takejto situácii sa od nás vyžaduje nájsť a realizovať čiastočné riešenie skôr, než obdržíme ďalšiu časť údajov nám nie úplne známeho vstupu.

Znázorníme to pomocou nasledujúceho príkladu. Predstavme si stredisko, ktoré poskytuje isté služby, napríklad pohotovostnú zdravotnú službu s 50 mobilnými lekármi. Každý lekár má k dispozícii vlastné motorové vozidlo. Keď niekto zavolá do centrály, centrála musí vyslať jedného z lekárov na miesto nehody alebo do bytu pacienta. Centrála pohotovostnej služby sa môže pokúsiť riadiť jazdy lekárov tak, aby sa optimalizovali isté para-

metre. Napríklad, centrála sa môže snažiť, aby

- bol čo najkratší priemerný čas čakania na pomoc,
- najdlhší čas čakania zo všetkých miest nehôd bol čo najkratší, alebo
- boli čo najmenšie náklady na benzín, alebo čo najnižší počet celkovo najazdených kilometrov.

Centrála má úplnú voľnosť v tom, ako sa bude rozhodovať, aby dosiahla svoj cieľ. Koho vyšle na najbližšie miesto nehody? Či lekára, ktorý práve skončil prácu na možno nie príliš vzdialenom mieste od miesta novej nehody alebo lekára čakajúceho v stredisku? Má lekár po skončení akcie zostať tam, kde je a čakať na ďalšie tiesňové volanie, alebo sa má vrátiť naspäť do strediska, alebo dokonca zaujať nejakú novú strategicky vhodnú vyčkávaciu pozíciu? Pri klasickom zadaní problému sú vopred známe všetky miesta a časy tiesňových volaní, ako aj dĺžky trvania každého zákroku a úlohou je nájsť riešenie vo forme plánu jázd jednotlivých lekárov, ktoré by optimalizovalo zvolené parametre. Samozrejme je nerealistické očakávať, že by sme v prípade pohotovostnej služby tieto informácie mohli vopred vedieť. Centrála netuší, kde a kedy dôjde k najbližšej nehode, a napriek tomu očakávame, že riadenie strediska pracuje podľa rozumnej stratégie, aj v prípade, že za týchto okolností nemožno zaručiť optimálne riešenie. Takto formulované úlohy nazývame **online problémy** a stratégie navrhnuté na hľadanie ich riešenia sa volajú **online algoritmy**.

Situácií, v ktorých potrebujeme online algoritmy je veľa a nie je ťažké si ich predstaviť. Napríklad riadenie taxislužby alebo policajnej stanice. Podobne je to aj v priemysle. Podniky musia často plánovať zadelenie svojich zamestnancov bez toho, aby tušili, koľko a ako štrukturovaných objednávok získajú v najbližšom čase. Problémy takého druhu sa volajú **problémy plánovania práce**[1]. Online problémy sú väčšinou veľmi ťažké, pretože budúcnosť môže byť plná schválností. Stiahneme lekára do strediska pohotovostnej služby a práve vtedy, keď sa vráti, príde tiesňové volanie od suseda predchádzajúceho pacienta. Nájsť dobrú za každých okolností optimalizujúcu stratégiu je často veľmi ťažké a niekedy je dokonca nemožné nájsť online stratégiu, ktorá by garantovala rozumné riešenia pre každý možný vývoj v budúcnosti. Umenie algoritmiky je vyskúmať, pre ktoré online úlohy existujú dobré online algoritmy, a pre ktoré online úlohy je beznádejné pokúšať sa urobiť dobré rozhodnutie pre neznámu budúcnosť. Zavše sa na prvý pohľad zdá beznádejné hrať proti nezná-

[1] po anglicky scheduling

mej budúcnosti, a napriek tomu existujú online algoritmy, ktoré nezávisle od vývoja v budúcnosti, garantujú riešenia blízke optimálnym. To sú ďalšie zázraky algoritmiky a cieľom tejto kapitoly je jeden z týchto divov predstaviť.

Nasledujúci odsek začneme úvodom do modelovania online problémov a vysvetlíme, ako môžeme merať kvalitu online algoritmov. Na ilustráciu ukážeme aj jeden konkrétny online problém, pri ktorom nie je možné prekabátiť neznámu budúcnosť.

V odseku 10.3. predstavíme jednu úlohu online plánovania pracovného procesu, a presvedčíme sa, že žiadna deterministická online stratégia nemôže zaručiť riešenie blízke optimálnemu. Potom z rukáva vytiahneme náhodou riadený online algoritmus, ktorý s vysokou pravdepodobnosťou garantuje riešenia veľmi blízke optimálnym. Ako zvyčajne, ukončíme kapitolu zhrnutím a predstavíme riešenie pre niektoré úlohy.

10.2 Meranie kvality online algoritmov a hra s prefíkaným protivníkom

Úlohy, ktoré študujeme v tejto kapitole, patria k **optimalizačným úlohám**. Každý prípad I nejakej optimalizačnej úlohy má potenciálne veľmi veľa riešení, ktoré voláme **prípustné riešenia pre I**. Rozpomeňme sa na úlohu obchodného cestujúceho (TSP). Pre každú úplne poprepájanú množinu n miest existuje obrovské množstvo $(n-1)!/2$ Hamiltonovských kružníc[2], ktoré zodpovedajú prípustným riešeniam. Úlohou ale nie je nájsť len jedno z prípustných riešení, ale také riešenie, ktoré má minimálne cestovné náklady alebo aspoň také, ktorého náklady sa veľmi nelíšia od optimálnych. Všetky online úlohy sú optimalizačné úlohy a cieľom je nájdenie prípustného riešenia pre úplne zadaný prípad úlohy, ktorý na začiatku poznáme len sčasti. V prípade pohotovostnej služby musíme (za predpokladu postačujúcej kapacity) riešiť všetky tiesňové situácie. Ak to dokážeme, našli sme prípustné riešenie, ktoré možno opísať postupnosťou inštrukcií mobilným lekárom. Je jedno, čoho mieru (cestovné náklady, časy čakania, alebo iné) chceme optimalizovať. Ak by sme vopred poznali budúcnosť so všetkými časmi a miestami tiesňových volaní, mohli by sme teoreticky[3]

[2]Hamiltonovské kružnice sú cesty, ktoré sa začínajú a končia v tom istom meste a všetkými ostatnými mestami prechádzajú práve raz.

[3]Slovom teoreticky mienime bez ohľadu na množstvo práce. Môže byť, že by sme to prakticky nezvládli kvôli priveľkej výpočtovej zložitosti.

vypočítať nejaké[4] optimálne riešenie. Bez toho, aby sme poznali budúcnosť, sa nám môže stať, že urobíme a realizujeme rozhodnutia, ktoré nám neskôr znemožnia efektívne riešiť nové prichádzajúce požiadavky. V tejto súvislosti si kladieme nasledujúcu principiálnu otázku:

> *Aký dobrý môže byť online algoritmus (ktorý nepozná budúcnosť) v porovnaní s algoritmom, ktorý od začiatku pozná celý prípad úlohy (budúcnosť)?*

Odpovede sa môžu líšiť od úlohy k úlohe. Ale aby sme mohli analyzovať túto otázku pre konkrétne úlohy, musíme najprv upresniť, čo znamená

> *„byť dobrý v porovnaní s algoritmom, ktorý pozná budúcnosť."*

Pri meraní kvality online algoritmov postupujeme podobne ako pri koncepte aproximačných algoritmov. Nech I je prípad uvedenej optimalizačnej úlohy U. Predpokladajme, že U je úloha minimalizácie. Nech

$$\mathbf{Opt}_{\boldsymbol{U}}(\boldsymbol{I})$$

označuje cenu optimálneho riešenia pre I. Nech A je online algoritmus pre U, ktorý vypočíta pre každý prípad úlohy I prípustné riešenie

$$\mathbf{Riešenie}_{\boldsymbol{A}}(\boldsymbol{I}).$$

Cenu tohto riešenia označíme ako

$$\mathbf{Cena}(\mathbf{Riešenie}_{\boldsymbol{A}}(\boldsymbol{I})).$$

Konkurenčnú kvalitu $\mathbf{Konk}_{\boldsymbol{A}}(\boldsymbol{I})$ algoritmu $\boldsymbol{A}$ pre prípad úlohy $\boldsymbol{I}$ definujeme ako číslo

$$\mathbf{Konk}_{\boldsymbol{A}}(\boldsymbol{I}) = \frac{\mathbf{Cena}(\mathbf{Riešenie}_{\boldsymbol{A}}(\boldsymbol{I}))}{\mathbf{Opt}_{\boldsymbol{U}}(\boldsymbol{I})}.$$

$\mathrm{Konk}_A(I)$ vyjadruje, koľkokrát horšie je riešenie $\mathbf{Riešenie}_A(I)$ algoritmu A vzhľadom na nejaké optimálne riešenie. Napríklad, keď $\mathrm{Opt}_U(I) = 100$ (optimálne riešenia pre I majú cenu 100) a $\mathrm{Cena}(\mathrm{Riešenie}_A(I)) = 130$ (riešenie vypočítané pomocou algoritmu A má cenu 130), znamená to

$$\mathrm{Konk}_A(I) = \frac{130}{100} = 1,3$$

[4] Píšeme nejaké, pretože môžu existovať viaceré optimálne riešenia.

teda, že vypočítané riešenie je 1,3-krát horšie ako najlepšie možné. Inými slovami Riešenie$_A(I)$ má o 30% vyššiu cenu ako optimálne riešenia pre I.

Teraz vieme určiť, aký dobrý je online algoritmus A na jednom prípade úlohy I. Celkovú kvalitu algoritmu A meriame z hľadiska záruky, ktorú je A schopný poskytnúť pre každý[5] prípad úlohy. Preto definícia kvality algoritmu A vyzerá nasledujúco:

Povieme, že A je **δ-konkurencieschopný online algoritmus** pre U, ak pre všetky prípady I úlohy U platí:

$$\text{Konk}_A(I) \leq \delta.$$

Ak platí $\text{Konk}_A(I) = 1,3$ pre všetky prípady I úlohy U (ak je teda A 1,3 -konkurencieschopný), znamená to, že A pre žiadny prípad úlohy nevypočíta riešenie, ktoré by bolo o viac ako 30% horšie od optimálneho. Pre veľa prípadov úlohy sa môže stať, že A vypočíta aj oveľa lepšie riešenie. Pre mnohé ťažké online úlohy môže byť už takáto záruka potešiteľná.

Úloha 10.1 Nech pre prípad úlohy I je $\text{Opt}_U(I) = 90$. Predpokladajme, že náš online algoritmus A vypočíta Riešenie$_A(I)$ s cenou. Cena(Riešenie$_A(I)$) = 135.

a) Určte konkurenčnú schopnosť algoritmu A pre I.
b) O koľko percent je Riešenie$_A(I)$ horšie ako optimálne riešenie pre I?

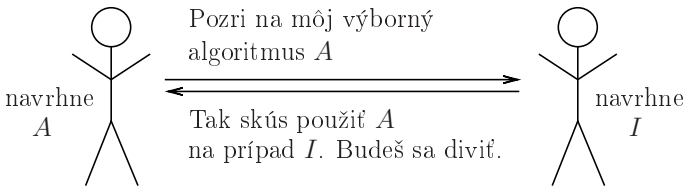
Úloha 10.2 Pre každý prípad problému I sme definovali čísla $\text{Opt}_U(I)$ a Cena(Riešenie$_A(I)$). Čo vyjadruje nasledujúce číslo

$$\frac{\text{Cena(Riešenie}_A(I)) - \text{Opt}_U(I)}{\text{Opt}_U(I)} \cdot 100 \ ?$$

Predpokladajme, že pre nejakú optimalizačnú úlohu U sme navrhli nejaký online algoritmus A a chceme analyzovať konkurencieschopnosť algoritmu A. To je zvyčajne veľmi ťažká matematická úloha. Poznáme množstvo online algoritmov pre rôzne úlohy, pre ktoré sa napriek dlhoročným pokusom nepodarilo zistiť ich konkurencieschopnosť. Obťažnosť tejto analýzy spočíva v tom, že hľadáme maximálnu hodnotu $\text{Konk}_A(I)$ vzhľadom na nekonečne veľa prípadov problému I.

Informatici používajú na analýzu konkurencieschopnosti online algoritmov symbolickú hru medzi dvoma hráčmi – navrhovateľom algoritmu a jeho protivníkom. Navrhovateľ algoritmov sa pokúša navrhovať online

[5] V informatike hovoríme často „v najhoršom prípade“.



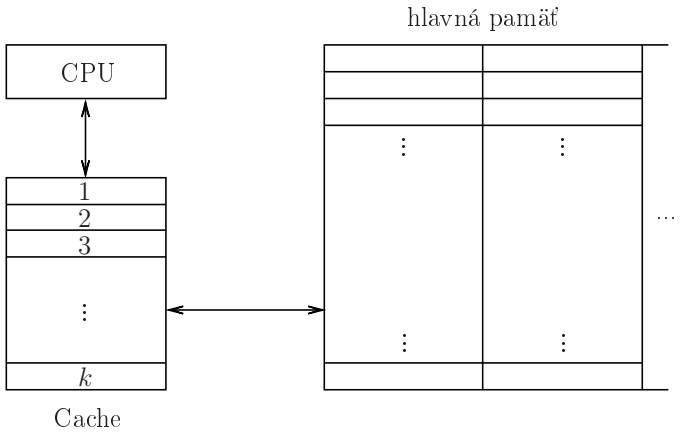
obr. 10.1

algoritmus a jeho protivník sa pokúša dokázať pre navrhnutý algoritmus prostredníctvom návrhu konkrétnych ťažkých prípadov problému, že navrhnutý algoritmus nie je z hľadiska konkurencieschopnosti dobrý (obr. 10.1).

V tejto hre môžeme navrhovateľa algoritmu považovať za na nadšeného optimistu, ktorý sa raduje z výsledkov svojej práce. Jeho protivníka môžeme pokladať za skeptika, ktorý má pochybnosti o všetkých produktoch navrhovateľa algoritmov a pokúša sa odôvodniť svoju skepsu. Dobré výskumné tímy potrebujú oba typy – pesimistov i optimistov. Takto prichádzajú na svet s nadšením nápady, ktoré budú dôsledne overené, opravené a v konečnom dôsledku vylepšené.

Pri analýze konkurencieschopnosti online algoritmov prisudzujeme protivníkovi veľkú dávku prefíkanosti alebo dokonca „zlomyseľnosti“. To je vec jeho výhodnej hráčskej situácie. Protivník pozná online algoritmus A a je schopný naplánovať budúcnosť tak, aby A nebol úspešný. Pretože mu je A známy, vie protivník, aké čiastkové riešenie A vypočíta pre každú časť (prefix) vstupu. Preto hru môžeme vidieť takto: Protivník ukáže kúsok budúcnosti a počká, čo s tým A urobí. Potom protivník určí ďalšiu časť vstupu. Potom, ako si pozrel (alebo sám vypočítal), ako na to A reaguje, zostaví ďalšie vstupné požiadavky. Vďaka tomuto postupu môže protihráč úspešne vodiť algoritmus A za nos. Ak sa to protihráčovi podarí, tak tým dokázal, že konkurencieschopnosť algoritmu A nie je vysoká (dobrá).

Nasledujúcim príkladom zabijeme jedným úderom tri muchy. Najprv ilustrujeme definíciu konkurencieschopnosti pomocou konkrétneho príkladu. Po druhé názorne predvedieme, ako môže prefíkaný protivník znemožniť každý online algoritmus pre danú úlohu. A po tretie pochopíme, že naozaj existujú úlohy, pre ktoré neexistuje úspešný online algoritmus.



obr. 10.2

Príklad 10.1 Stránkovanie[6]

Stránkovanie je aktuálna úloha, ktorá sa musí nepretržite riešiť v každom počítači. V počítači máme minimálne dva druhy pamäte, jednu malú a jednu veľkú. Malá časť sa volá **cache**[7] a v tejto pamäti máme k údajom super rýchly prístup. Veľká pamäť sa volá **hlavná pamäť** a je podstatne väčšia ako cache (obr. 10.2).

Prístup k údajom v hlavnej pamäti je pomalý. V podstate je to dokonca tak, že počítač môže priamo spracovávať len údaje, ktoré sú uložené v pamäti cache. Ak chce počítač nahliadnuť do údajov, alebo spracovávať údaje, ktoré nie sú v cache, musí najprv tieto údaje do cache preniesť, a potom ich môže odtiaľ čítať. Preto sa počítač „snaží“ mať uložené v cache všetky aktuálne údaje, ktoré sa budú v najbližšom čase spracovávať. To sa ale nedá zaručiť, pretože počítač nepozná budúcnosť, a teda nevie, ku ktorým údajom bude chcieť používateľ mať v najbližšom čase prístup. Počítač dostáva len z času na čas isté požiadavky, poskytnúť k nahliadnutiu alebo spracovať isté údaje. Ak tie údaje nie sú v cache, musí ich tam pretransportovať. Pretože cache je zvyčajne plná, musí počítač najprv nejakú časť údajov v cache vymazať alebo z cache preniesť do hlavnej pamäti. A teraz prichádza priestor pre online stratégie. Ktoré údaje máme vybrať (odstrániť) z cache? Samozrejme tie, ktoré nebudeme pri najbližších požiadavkách potrebovať!

Modelujme vzniknutú situáciu presnejšie. Obidve pamäte sú rozdelené

[6] v angličtine „paging“
[7] v slovenčine rýchla vyrovnávacia pamäť

do dátových blokov, ktoré nazývame **stránky**. cache môže obsahovať najvýš k stránok pre nejaké pevné číslo k, a typicky obsahuje práve k stránok, teda je plná. Hlavná pamäť obsahuje všetky potrebné údaje, takže tam môže byť uložené obrovské množstvo údajov. Medzi cache a hlavnou pamäťou sa dajú „prenášať“ len celé stránky. Celé to pripomína hru s hrubou knihou, v ktorej môžeme mať otvorených naraz najviac k strán. Ak chceme vidieť nejakú novú stranu, musíme najprv jednu z k otvorených strán zavrieť. Z toho dôvodu voláme túto úlohu stránkovanie (listovanie).

Príklad prípadu stránkovanie je napr. postupnosť:

$$I = 3, 107, 30, 1201, 73, 107, 30$$

Tento prípad stránkovania zodpovedá požiadavke čítať stránky s číslami 3, 107, 30, 1201, 73, 107 a 30, v tomto poradí, jednu po druhej, stránky 107 a 30 si pozrieme dokonca dvakrát. Stránku 107 najprv v druhom a potom v šiestom časovom odseku. To znamená, že by bolo chybou vyhodiť po prvom pozretí z cache stránku 107, pretože by sme ju v takom prípade museli pred šiestym časovým okamihom zase preniesť nazad do cache.

Vzhľadom na to, že stránkovanie prezentujeme ako úlohu minimalizácie, potrebujeme merať cenu jednotlivých riešení. Časy potrebné na prenos údajov medzi hlavnou pamäťou a cache a na prístup k údajom v cache sú natoľko rozdielne, že stanovíme:

- cenu 0 za prístup na stránku v cache a
- cenu 1 za prenos údajov z hlavnej pamäte do cache.

Predstavme si nasledujúcu konkrétnu situáciu. Máme cache veľkosti 3 ($k = 3$), ktorá obsahuje práve 3 stránky 5, 30 a 107. Teraz dostaneme už spomínaný prípad úlohy I = 3, 107, 30, 1201, 73, 107, 30. Nasledujúce optimálne riešenie má cenu 3. Počítač najprv prenesie stránku 3 z hlavnej pamäte do cache a súčasne s tým stránku 5 z cache do hlavnej pamäte. Túto výmenu stránok 3 a 5 medzi cache a hlavnou pamäťou budeme zapisovať v nasledujúcom texte symbolicky ako

$$5 \leftrightarrow 3.$$

Ďalšie požadované strany 107 a 30 sú už v cache k dispozícii, a preto nedochádza k žiadnemu prenosu údajov medzi cache a hlavnou pamäťou. Potom prenesieme stranu 1201 z hlavnej pamäti na miesto stránky 3 v cache. Strany 30 a 107 z cache neodstránime, lebo vieme, že ich v budúcnosti ešte budeme potrebovať. V piatom kroku vymeníme stránku

čas	0	1	2	3	4	5	6	7
operácia		$5 \leftrightarrow 3$	•	•	$3 \leftrightarrow 1201$	$1201 \leftrightarrow 73$	•	•
cache	5	3	3	3	1201	73	73	73
	30	30	30	30	30	30	30	30
	107	107	107	107	107	107	107	107
načítané		3	107	30	1201	73	107	30

Tabuľka 10.1

1201 za stránku 73. Posledné dve požiadavky 107 a 30 môžeme realizovať bez komunikácie s hlavnou pamäťou. Hore opísané riešenie môžeme v krátkosti symbolicky zapísať ako

$$5 \leftrightarrow 3, \bullet, \bullet, 3 \leftrightarrow 1201, 1201 \leftrightarrow 73, \bullet, \bullet,$$

kde • symbolizuje časový interval, v ktorom neprebieha žiadna komunikácia medzi cache a hlavnou pamäťou. Označenie $a \leftrightarrow b$ symbolizuje výmenu stránky a z cache za stránku b z hlavnej pamäti. Tabuľka 10.1 znázorňuje obsah cache v uvažovaných ôsmich časových intervaloch.

To, že naše riešenie je optimálne, môžeme tvrdiť vďaka tomu, že na začiatku nie sú stránky $3, 1201$, a 73 v cache. Takže je jasné, že pri požiadavkách $I = 3, 107, 30, 1201, 73, 107, 30$ tam niekedy musia byť prenesené, a teda sú nevyhnutné minimálne tri akcie prenosu údajov z hlavnej pamäte.

Úloha 10.3 Nájdite optimálne riešenia pre nasledujúce prípady úlohy stránkovania:

a) $k = 3$, cache obsahuje $1, 2, 3$ a $I = 7, 9, 3, 2, 14, 8, 7$
b) $k = 5$, cache obsahuje $1, 101, 1001, 1002, 9$ a $I = 1002, 7, 5, 1001, 101, 3, 8, 1, 1002$

Na riešenie daného prípadu môže existovať veľa riešení. Ak požadujeme stránku, ktorá nie je v cache, tak máme k možností, ktorú stránku z cache vymeníme za požadovanú novú stránku. S dĺžkou postupnosti požiadaviek na čítanie stránok exponenciálne rastie počet možných riešení a situácia sa môže stať neprehľadnou. Napriek tomu vždy existuje možnosť nájsť (aj keď s nemalým úsilím) optimálne riešenie. Ak ale stojíme pred stránkovaním ako online úlohou, mení to podstatne situáciu. Požiadavky prichádzajú osobitne jedna po druhej. Až po splnení prijatej požiadavky, vďaka čomu sme potenciálne odstránili z cache nejakú stránku, dostaneme ďalšiu požiadavku. Protivník má v tejto hre príliš veľa priestoru

a môže sa správať značne zlomyselne. V ďalšej požiadavke si zvolí vždy tú stránku, ktorú sme práve z pamäti odstránili. Takto dosiahne stav, keď každá jeho požiadavka zapríčiní výmenu stránok medzi cache a hlavnou pamäťou. Tým sa maximalizuje cena riešenia, lebo väčšia ako počet požiadaviek ani nemôže byť.

Ukážme si to na konkrétnom príklade. Položme $k = 3$ a predpokladajme, že cache obsahuje stránky 1, 2 a 3. Protivník požaduje stránku 4. Jednu stránku z cache musíme teda odstrániť. Predpokladajme, že uvedená online stratégia odstráni stránku 2, t.j. vykoná akciu $2 \leftrightarrow 4$. V takom prípade žiada protivník stránku 2. Tá ale už nie je v cache, a teda ju tam musíme preniesť. Predpokladajme, že online algoritmus realizuje $4 \leftrightarrow 2$. Následne požaduje protivník stránku 4. Ak sa online stratégia rozhodne pre $1 \leftrightarrow 4$, tak v ďalšom protivník požaduje stránku 1, čo možno splniť akciou $4 \leftrightarrow 1$. Takto vytvoril protivník prípad úlohy

$$4, 2, 4, 1$$

a online algoritmus vyprodukoval riešenie

$$2 \leftrightarrow 4, 4 \leftrightarrow 2, 1 \leftrightarrow 4, 4 \leftrightarrow 1$$

s maximálnou možnou cenou 4.

Pokiaľ ale poznáme od začiatku celé zadanie prípadu problému $4, 2, 4, 1$, tak navrhneme riešenie

$$3 \leftrightarrow 4, \bullet, \bullet, \bullet,$$

ktoré má cenu len 1.

Úloha 10.4 Uvažujme nasledujúci online algoritmus. Ak treba odstrániť stránku z cache, tak vezmeme vždy stránku s najmenším číslom. Položme $k = 4$ a predpokladajme, že na začiatku obsahuje cache stránky $1, 3, 5, 7$. Hrajte úlohu protivníka a navrhnite prípad problému pozostávajúci z 10 požiadaviek tak, aby táto online stratégia viedla k riešeniu s cenou 10, i keď je možné riešenie s cenou 1.

Úloha 10.5 Predstavme si online algoritmus, ktorý v prípade potreby vyberie z cache tú stránku, ktorá bola doteraz najmenej požadovaná. Ak existujú viaceré takéto stránky, odstráni spomedzi najmenej požadovaných stránok tú s najvyšším číslom. Hrajte úlohu protivníka s touto stratégiou a nasledujúcimi štartovacími situáciami a podmienkami.

a) $k = 4$, cache obsahuje $1, 2, 3, 4$, prípad I má pozostávať z postupnosti štyroch požiadaviek a $\mathrm{Opt}_{stránkovanie}(I) = 1$.

b) $k = 5$, cache obsahuje $1, 7, 103, 5, 9$ a pre navrhnutý ťažký prípad stránkovania o 10 požiadavkách má platiť $\text{Opt}_{stránkovanie}(I) = 2$.

Zovšeobecnime naše doterajšie skúsenosti a dokážme, že pre každý online algoritmus A pre stránkovanie cache s k stránkami existuje prípad stránkovania I taký, že

$$\text{Konk}_A \geq k,$$

a teda neexistuje žiadna online stratégia, ktorá by poskytovala uspokojivé riešenia pre všetky prípady stránkovania.

Predpokladajme, že v cache, ktorá má veľkosť k stránok, máme stránky $1, 2, 3, 4, \ldots, k$. Nech A je ľubovoľný online algoritmus pre úlohu stránkovania. Protivník začne konštruovať prípad úlohy požiadavkou $k + 1$. Pretože stránka $k + 1$ v cache nie je, A ju musí vymeniť za jednu zo stránok $1, 2, \ldots, k$. Nech A urobí rozhodnutie:

$$s_1 \leftrightarrow k + 1$$

pre nejaké s_1 z $\{1, 2, \ldots, k\}$. Teraz cache obsahuje stránky:

$$1, 2, \ldots, s_1 - 1, s_1 + 1, \ldots, k, k + 1 \ .$$

V tomto prípade pokračuje protivník s požiadavkou s_1. Stránka s_1 nie je v cache, a tak musí A zase nariadiť výmenu stránok medzi cache a hlavnou pamäťou. Nech A spraví rozhodnutie:

$$s_2 \leftrightarrow s_1$$

pre nejaké s_2 z $\{1, 2, \ldots, k, k + 1\} - \{s_1\}$. Teraz obsahuje cache stránky $1, 2, \ldots, s_2 - 1, s_2 + 1, \ldots, k + 1$ (presnejšie všetky stránky $i \in \{1, 2, \ldots, k + 1\} - \{s_2\}$). Prirodzene, protivník teraz požaduje práve tú chýbajúcu stránku s_2. Týmto spôsobom môžeme pokračovať tak dlho, kým protivník nevytvorí prípad úlohy:

$$I_A = k + 1, s_1, s_2, \ldots, s_{k-1}$$

dĺžky k a s obťažnosťou:

$$\text{Cena}(\text{Riešenie}_A(I_A)) = k.$$

Všimnite si, že pre rôzne online stratégie vznikajú rôzne prípady problému. Tvrdíme, že platí

$$\text{Opt}_{stránkovanie}(I_A) = 1.$$

Pokúsme sa to zdôvodniť. Ak od začiatku poznáme celý prípad I_A, môžeme postupovať takto: Hodnoty $s_1, s_2, \ldots, s_{k-1}$ sú všetky z $\{1, 2, \ldots, k+1\}$ a je ich presne $k-1$. To znamená, že existuje číslo j z $\{1, 2, \ldots, k\}$, ktoré sa nenachádza medzi hodnotami $s_1, s_2 \ldots, s_{k-1}$, čiže stránka j v I_A nebude nikdy požadovaná na nahliadnutie. Preto pri prvej požiadavke na stránku $k+1$ zvolíme akciu

$$j \leftrightarrow k+1.$$

Po tejto akcii nie je potrebný žiadny prenos údajov medzi cache a hlavnou pamäťou, pretože všetky stránky $s_1, s_2 \ldots s_{k-1}$, ktoré zodpovedajú nasledujúcim $k-1$ požiadavkám prípadu úlohy I_A, sú už v cache. Zo stránok $\{1, 2, \ldots k+1\}$ chýba v cache len stránka j, ktorá ale nie je v I_A nikdy požadovaná. (Ak napr. $k = 4$ a $I_A = 5, 3, 1, 4$, tak $j = 2$, a tak je riešenie $2 \leftrightarrow 5, \bullet, \bullet, \bullet$ optimálne). Pretože $\mathrm{Opt}_{stránkovanie}(I_A) = 1$, dostávame

$$\mathrm{Konk}_A(I_A) = \frac{\mathrm{Cena}(\mathrm{Riešenie}_A(I_A))}{\mathrm{Opt}_{stránkovanie}(I_A)} = \frac{k}{1} = k,$$

pre každý uvažovaný online algoritmus A. Na základe toho usudzujeme, že neexistuje žiadny δ-konkurenčne schopný online algoritmus pre úlohu stránkovania v cache veľkosti k, pre $\delta < k$.

Ako riešime tento problém v praxi? Vzhľadom na obťažnosť analýzy sa tým nechceme podrobne zaoberať. Základná myšlienka je navrhnúť online algoritmus, ktorý produkuje dobré výsledky aspoň na typických, keď už nie vo všetkých prípadoch stránkovania. Vďaka rozsiahlemu skúmaniu prípadov úloh z praxe vieme, že stránky, ktoré boli požadované častejšie v poslednom čase, majú vyššiu pravdepodobnosť, že budú znova požadované. Čo rozumieme pod „v poslednom čase“ a akú rolu presne zohráva experimentálna početnosť rôznych požiadaviek, sa tu nepokúsime presne špecifikovať. Okrem toho je tu ešte možnosť získať online algoritmy s náhodným riadením, ktorá môže byť pre riešenie úlohy stránkovania veľmi užitočná.

10.3 Randomizovaný online algoritmus pre jednu úlohu plánovania

V tomto odseku chceme ukázať, že je možné objaviť randomizované algoritmy, ktoré i v zdanlivo beznádejných situáciách dokážu zaručiť kvalitné

riešenia blízke optimálnym. To znamená, že existujú ťažké úlohy, v ktorých napriek očakávaniu sa môžeme rozhodovať bez znalosti budúcnosti skoro tak dobre, ako niekto, kto je schopný predpovedať budúcnosť.

Aby sme ostali názorní a predišli príliš komplikovaným matematickým analýzam a argumentom, predstavme si jednoduchú verziu úlohy plánovania práce. Predpokladajme, že máme podnik s n rôznymi pracoviskami. Na každom pracovisku je zariadenie, ktoré je schopné vykonávať isté pracovné úkony buď samostatne, alebo riadením nejakého pracovníka. Do podniku prichádzajú zákazníci s rôznymi objednávkami. V objednávke je uvedené, ktoré pracoviská a v akom poradí zákazník potrebuje, aby jeho objednávka bola úspešne vybavená. V zjednodušenej verzii predpokladáme, že každá objednávka potrebuje všetky stanice a každá stanica je požadovaná na rovnako dlhý čas (napr. 10 minút). Jediné, čo v tejto jednoduchej verzii úlohy môže zákazník zvoliť, je, v akom poradí musia byť použité pracoviská. Napríklad pre $n = 5$ má podnik 5 staníc S_1, S_2, S_3, S_4 a S_5. Objednávka

$$A = (1, 3, 5, 4, 2)$$

znamená, že zákazník požaduje pre spracovanie svojej objednávky zariadenia (pracoviská) v poradí

$$S_1, S_3, S_5, S_4 \text{ a } S_2.$$

Úlohou podniku je objednávku podľa možnosti čo najskôr vybaviť. Ak je objednávka A jedinou, úloha je pre podnik hravo zvládnuteľná v priebehu 5 časových jednotiek (jedna časová jednotka pre jedno pracovisko). V prvej časovej jednotke pracuje pracovisko S_1, v druhej stanica S_3, v tretej S_5, vo štvrtej S_4 a v poslednej piatej časovej jednotke ukončí prácu na objednávke pracovisko S_2. Úloha spočíva v tom, že v hre môžu byť viacerí zákazníci, ktorí si navzájom môžu konkurovať pri spracovaní svojich zákaziek. Budeme sa zaoberať najjednoduchším variantom úlohy, kde sa vyskytujú len dve objednávky od dvoch zákazníkov. Podnik sa usiluje minimalizovať čas na úplné spracovanie oboch objednávok. Predstavme si napríklad pre $n = 4$ nasledujúce dve objednávky

$$\begin{aligned} A_1 &= (1, 2, 3, 4) \\ A_2 &= (3, 2, 1, 4). \end{aligned}$$

Podnik môže riešiť objednávky nasledujúcim spôsobom: V prvej časovej jednotke spracúva obe objednávky **paralelne**, čo znamená, že prvé

pracovisko S_1 pracuje na prvej úlohe objednávky A_1 a tretie pracovisko S_3 pracuje na prvej úlohe objednávky A_2. Po uplynutí prvej časovej jednotky sú obe objednávky hotové s prvou úlohou a potrebujú obe pracovať na pracovisku S_2. To ale nemôžeme realizovať súčasne, pretože na jednom pracovisku môže byť naraz spracovávaná len jedna úloha. Nech sa podnik rozhodne spracovať najprv druhú úlohu prvej objednávky A_1 na požadovanom pracovisku S_2. Vzhľadom na to, že úlohy musia by spracované v danom poradí, nebude sa v druhom časovom úseku pracovať na druhej objednávke A_2. Kvôli tomu má A_2 **oneskorenie** veľkosti jednej časovej jednotky. Po uplynutí druhej časovej jednotky sú splnené prvé dve úlohy objednávky A_1 a na objednávke A_2 bola doteraz vykonaná len prvá úloha. (tab. 10.2). Teraz požaduje A_1 pracovisko S_3 a A_2 pracovisko S_2. Obe požiadavky možno splniť súčasne, a teda sa realizujú. Po tretej časovej jednotke ostáva len požiadavka objednávky A_1 na pracovisko S_4 a na objednávky A_2 treba splniť ešte požiadavky 1,4. Vidíme, že podnik vo štvrtej časovej jednotke môže súčasne spracovať A_1 na S_4 a A_2 na S_1. Tým je objednávka A_1 po štvrtej časovej jednotke úplne spracovaná bez akéhokoľvek oneskorenia. V piatej časovej jednotke môže potom podnik dokončiť na pracovisku S_4 prácu na objednávke A_2. Po piatich časových jednotkách sú obe objednávky A_1 a A_2 vybavené a naše riešenie je optimálne, pretože neexistuje riešenie, ktoré by tieto objednávky dokázalo vybaviť v kratšom čase.

Riešenie v kratšom čase je nemožné preto, lebo spracovanie A_1 a A_2 v priebehu 4 časových jednotiek znamená, že v každej časovej jednotke musia byť súčasne spracované obidve objednávky. Na začiatku to znamená konkrétne spracovanie A_1 na S_1 a spracovanie A_2 na S_3. Tým je nevyhnutný **konflikt** rovnakých požiadaviek oboch zákaziek na použitie pracoviska S_2 v druhej časovej jednotke, ktorému sa nemôžeme nijako vyhnúť. Pre nemožnosť spracovať obe objednávky súčasne musí jedna z nich čakať počas druhej časovej jednotky, a teda nemôže byť vybavená skôr ako po piatich časových jednotkách.

Nasledujúca tabuľka znázorňuje priebeh vyššie opísaného spracovania objednávok A_1 a A_2. Každý stĺpec zodpovedá jednej časovej jednotke a ukazuje, ktoré pracoviská aktívne pracujú na ktorej úlohe. Z pohľadu riadkov môžeme pozorovať, do akej miery pokročila práca na objednávkach A_1 a A_2 po uplynutí jednotlivých časových jednotiek.

Úloha 10.6 Rozoberme si situáciu, keď sa podnik rozhodne v prvej časovej jednotke spracovávať len A_1 na pracovisku S_1 a nechá A_2 čakať, i keď požadované pracovisko S_3 má nevyužité k dispozícii. Dá sa v takomto prípade ešte dosiah-

časové jednotky	1	2	3	4	5
A_1	S_1	S_2	S_3	S_4	
A_2	S_3		S_2	S_1	S_4

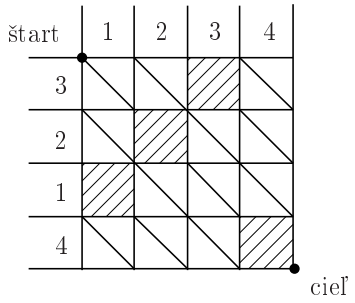
Tabuľka 10.2

nuť, aby bola práca na oboch objednávkach ukončená v rámci piatich časových jednotiek? Opíšte vaše priradenie pracovísk objednávkam v čase, podobne ako v tab. 10.2.

Úloha 10.7 Predpokladajme, že máme podnik s $n = 6$ pracoviskami a dve objednávky $A_1 = (1, 2, 3, 4, 5, 6)$ a $A_2 = (1, 3, 2, 6, 5, 4)$. Koľko časových jednotiek potrebujete na úplné vybavenie objednávok? Prezentujte svoje riešenie vo forme tabuľky tab. 10.2.

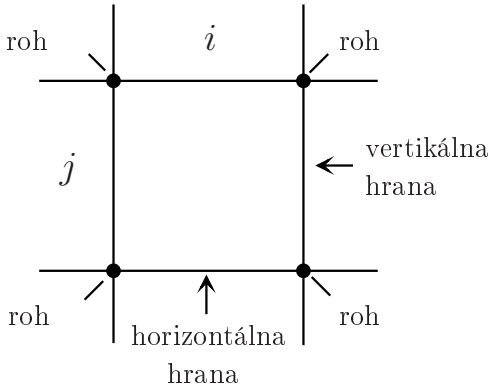
Úloha rozdeľovať prácu sa dá predstaviť graficky, čo znázorňuje hľadanie dobrých riešení, a tým ich aj uľahčuje. Pre podnik s n pracoviskami si nakreslíme najprv tabuľku $n \times n$. Pre dve objednávky A_1 a A_2 napíšeme postupnosť požadovaných pracovísk objednávky A_1 do stĺpcov a postupnosť požadovaných pracovísk objednávky A_2 do riadkov. Na obrázku 10.3 vidíme tabuľku 4×4 pre objednávky:

$$A_1 = (1, 2, 3, 4) \text{ a } A_2 = (3, 2, 1, 4) \ .$$



obr. 10.3

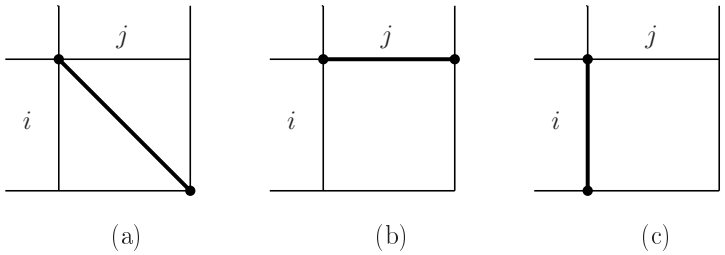
Políčka, v ktorých sa stretávajú rovnaké požiadavky (ako napr. priesečník prvého riadku s tretím stĺpcom zodpovedá súčasnej požiadavke oboch objednávok použiť pracovisko S_3, alebo priesečník druhého riadku s druhým stĺpcom zodpovedá súčasnej požiadavke práce na pracovisko S_2) nazývame **prekážky** a označíme ich vyšrafovane. Pozorujeme, že počet prekážok je práve n, pretože pre každú stanicu S_i $(i = 1, \ldots n)$ existuje



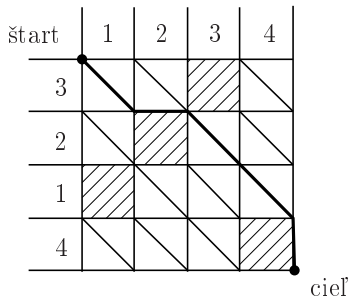
obr. 10.4

práve jeden riadok a jeden stĺpec označený číslom i. Takže na obrázku 10.3 vidíme presne štyri prekážky. Vo všetkých voľných políčkach bez prekážok (políčka s rôznymi požiadavkami objednávok A_1 a A_2) nakreslíme spojnicu z horného ľavého vrchola (rohu) políčka do dolného pravého vrcholu (rohu) políčka a nazveme ju **diagonálna hrana** (obr. 10.3). Body, kde sa pretínajú horizontálne a vertikálne čiary oddeľujúce jednotlivé riadky a stĺpce nazývame vrcholmi (obr. 10.4).

Úsečky, ktoré spájajú susedné vrcholy, nazývame **hrany**. Hrany, ktoré ležia na horizontálnych čiarach (oddelujúcich riadky), nazývame **horizontálne hrany** (obr. 10.4). Analogicky nazývame vertikálne orientované úsečky medzi dvoma susednými vrcholmi **vertikálne hrany**. Hľadanie riešenia pre spracovanie objednávok A_1 a A_2 zodpovedá v tejto grafickej reprezentácii hľadaniu cesty z horného ľavého rohu tabuľky $n \times n$ (označený na obr. 10.3 ako štart) do dolného pravého rohu tabuľky (označené na obr. 10.3 ako cieľ). Cesta prebieha postupne z vrchola do susedného vrchola.



obr. 10.5



obr. 10.6

Ak prechádzame diagonálnou hranou políčka na priesečníku i-teho riadku a j-teho stĺpca (obr. 10.5(a)), tak to zodpovedá súčasnému spracovaniu objednávok A_1 a A_2 na požadovaných pracoviskách S_i a S_j. Použitie horizontálnej hrany (obr. 10.5(b)) zodpovedá spracovaniu objednávky A_1 na požadovanom pracovisku S_j a čakaniu objednávky A_2, ktorá v tomto časovom intervale nie je spracovaná a kvôli tomu sa oneskorí o jednu časovú jednotku. Ísť po vertikálnej hrane (obr. 10.5(c)) zodpovedá spracovaniu A_2 na požadovanom pracovisku S_i a čakaniu A_1 na ďalšie spracovanie. Na obrázku 10.6 je znázornené riešenie z tab. 10.2, ktoré zodpovedá plánu pre prípad úlohy z 10.3. Jediná horizontálna hrana použitá v riešení zodpovedá obchádzaniu prekážky v políčku (2, 2). Jediná vertikálna hrana je použitá v poslednom kroku na dosiahnutie cieľa, keď už je objednávka A_1 vybavená a dokončujeme len objednávku A_2.

Cena riešenia (počet potrebných časových jednotiek na úplné spracovanie A_1 a A_2) je presne dĺžka cesty zo štartu do cieľa počítaná ako počet prejdených hrán. Na obrázku 10.6 pozostáva cesta zo štartu do cieľa z 5 hrán a i-ta hrana cesty zodpovedá presne priradeniu pracoviska v i-tej časovej jednotke v tab. 10.2. Ak poznáme od začiatku úplne objednávky A_1 a A_2, môžeme nájsť optimálne riešenie priradenia pracovísk jednotlivým objednávkam v jednotlivých časových intervaloch veľmi rýchlo vďaka efektívnym algoritmom na hľadanie najkratšej cesty zo štartu do cieľa.

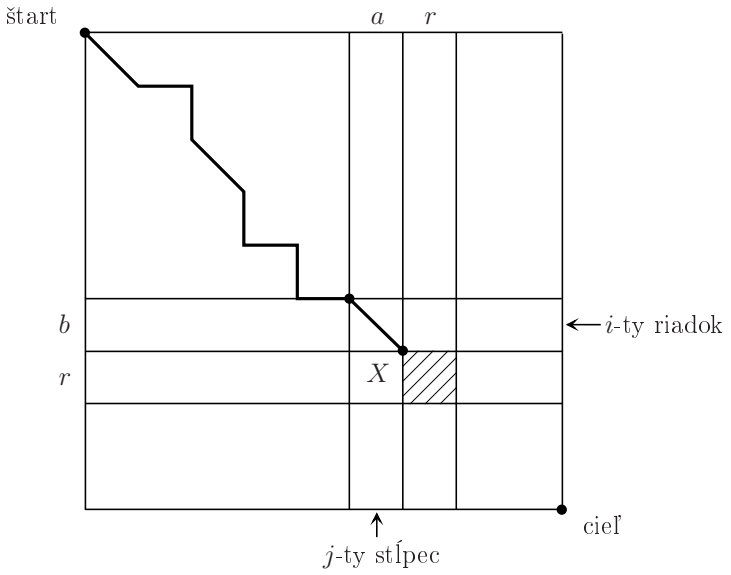
Úloha 10.8 Predstavte si objednávky $A_1 = (6, 5, 4, 3, 2, 1)$ a $A_2 = (4, 5, 6, 2, 3, 1)$ pre podnik so šiestimi pracoviskami! Načrtnite zodpovedajúce grafické zobrazenie tohto prípadu úlohy so šiestimi prekážkami. Nájdite jednu z najkratších ciest (môže ich existovať viacej) zo štartu do cieľa a zostavte k tejto ceste zodpovedajúcu tabuľku, ktorá opisuje priradenia pracovísk objednávkam podobne ako ste to urobili v tab. 10.2.

Vďaka grafickému zobrazeniu vidíme, že túto optimalizačnú úlohu vieme

vyriešiť ľahko a rýchlo. Situácia sa ale zmení, ak budeme túto úlohu pokladať za online úlohu. Na začiatku pozná podnik len prvú požiadavku objednávky A_1 a prvú požiadavku (úlohu) objednávky A_2, napr. $A_1 = (3, \ldots)$ s $A_2 = (5, \ldots)$. Zvyšok prípadu úlohy je podniku zatiaľ neznámy. Ak podnik vybaví súbežne prvé čiastkové úlohy objednávok na pracoviskách S_3 a S_5, objavia sa nasledovné úlohy objednávok A_1 a A_2. Pravidlom je, že až po rozhodnutí a spracovaní daných čiastkových úloh objednávok A_1 a A_2 sa podnik dozvie, aké ďalšie čiastkové úlohy objednávok A_1 a A_2 sú na rade.

Pri tomto scenári môže hra medzi navrhovateľom online algoritmu s jeho protivníkom vyzerať takto: Navrhovateľ algoritmu navrhne nejakú online stratégiu a jeho protivník navrhne prípad úlohy, ktorý je ťažký pre zvolenú stratégiu. Stratégia protivníka v tejto hre je veľmi jednoduchá a uvidíme, že zlomyselnejšiu si naozaj ani nemožno predstaviť. Pre každý algoritmus zostrojí protivník taký prípad úlohy, že danému algoritmu neostane nič iného, ako minimálne v každom druhom kroku použiť nejakú nediagonálnu hranu. To znamená, že v každom druhom kroku dôjde k oneskoreniu pri vybavovaní na jednej z objednávok. Ako to môže protivník dosiahnuť?

Predpokladajme (obr. 10.7), že posledný krok algoritmu použil nejakú diagonálnu hranu.



obr. 10.7

Ak sa nachádzame vo vrchole X (obr 10.7) v dolnom pravom rohu prie-

sečníka i-teho riadku a j-teho stĺpca, znamená to, že prvých j úloh objednávky A_1 a prvých i úloh objednávky A_2 je už spracovaných. Teraz si môže protivník zvoliť $(j+1)$-vú úlohu objednávky A_1 a $(i+1)$-vú úlohu objednávky A_2. Protivník zvolí tú istú požiadavku na pracovisko S_r, ktoré doteraz ešte nebolo použité. Takto vychádzajúc z X leží pred nami prekážka a nám neostáva nič iné, ako pokračovať nejakou nediagonálnou hranou (Musíme si vybrať z dvoch možností znázornených na obr. 10.5(b) a obr. 10.5(c)).

Ak navrhnutý online algoritmus po diagonálnom kroku dosiahne okraj tabuľky (obr. 10.8), tak neexistuje tak či tak nijaká možnosť používať až do konca cesty len diagonálne hrany. Preto môže protivník zvoliť zvyšné úlohy ešte nedokončenej objednávky (na obr. 10.8 je to objednávka A_2) v ľubovoľnom poradí.

Čo môžeme vydedukovať z tejto úvahy? Pre každý online algoritmus A nájde protivník prípad problému x_A, na ktorom online algoritmus A minimálne v každom druhom kroku použije nediagonálnu hranu. To znamená, že A musí spôsobiť minimálne v každom druhom kroku oneskorenie pri spracovaní jednej z objednávok. Ak máme m staníc, musíme počítať s $m/2$ oneskoreniami, ktoré sú rozdelené na obe objednávky. Z toho vyplýva, že aspoň jedna z objednávok musí mať minimálne $m/4$ oneskorení. Takže čas spracovania prípadu úlohy x_A pomocou riešenia vypočítaného algoritmom A je aspoň

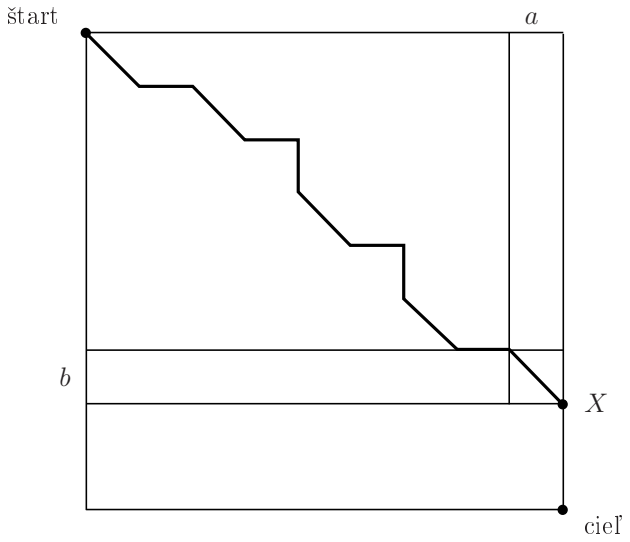
$$m + m/4 \ ,$$

pretože najkratšie spojenie zo štartu do cieľa pozostáva z m diagonálnych hrán a k minimálnemu času musíme ešte pripočítať maximum oneskorení na jednotlivých objednávkach.

Ilustrujme hru protivníka proti jednej konkrétnej online stratégii.

Príklad 10.2 Predstavme si nasledovnú online stratégiu A na hľadanie cesty zo štartu do cieľa v tabuľke znázorňujúcej dve objednávky.

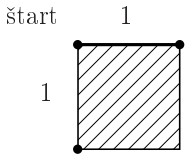
1. Ak je možné použiť diagonálnu hranu, tak ju použi.
2. Ak stojíš pred prekážkou (t.j. bez možnosti ísť po diagonálnej hrane), zvoľ si tú z dvoch možných hrán (jednej horizontálnej a jednej vertikálnej), ktorá vedie bližšie k hlavnej diagonále celého poľa medzi štartom a cieľom. Ak sú obe možnosti rovnako dobré, zvoľ si horizontálnu hranu.
3. Ak sa nachádzaš na pravom okraji tabuľky, použi na dosiahnutie cieľa vertikálne hrany.



obr. 10.8

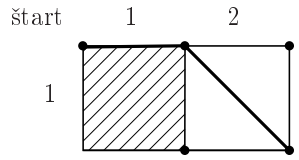
4. Ak sa nachádzaš na spodnom okraji tabuľky, použi na dosiahnutie cieľa horizontálne hrany.

Pre tento online algoritmus A skonštruuje protivník ťažký prípad plánovania $x_A = (A_1, A_2)$ takto: Konštrukcia sa začína s $A_1 = 1, \ldots$ a $A_2 = 1, \ldots$. Takto leží prekážka hneď na štarte (obr. 10.9) a A ju musí obísť. Podľa bodu 2 algoritmu A sú obe možnosti vertikálna i horizontálna rovnako dobré a tak ide A po horizontálnej hrane.



obr. 10.9

Po tomto kroku musí protivník sformulovať druhú úlohu (požiadavku) objednávky A_1. Teraz nemá protivník nijakú možnosť položiť prekážku, pretože v každom riadku existuje práve jedna prekážka, a tú už protivník do prvého riadku uložil. Preto je jedno, ktoré pracovisko si protivník vyberie. Nech je to pracovisko S_2, a teda nech $A_1 = (1, 2, \ldots)$. Teraz môže A pokračovať po diagonálnej hrane (obr. 10.10) a spracuje tým súčasnú úlohu 1 objednávky A_1 a úlohu 2 objednávky A_2.

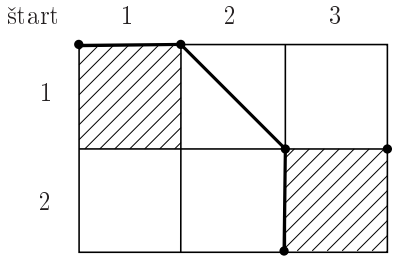


obr. 10.10

Teraz sú splnené všetky doteraz špecifikované požiadavky a protivník môže sformulovať nasledovné požiadavky oboch objednávok A_1 a A_2. To znamená, že protivník môže znova položiť prekážku (použitou diagonálnou hranou dosiahla cesta nový riadok a nový stĺpec, v ktorom neležia ešte nijaké prekážky). Nech protivník pokračuje v konštrukcii prípadu úlohy (A_1, A_2) nasledovne:

$$A_1 = (1, 2, \mathbf{3}, \ldots) \text{ a } A_2 = (1, \mathbf{3}, \ldots) \ .$$

Takže algoritmus A narazil zasa na prekážku (obr. 10.11). Podľa bodu 2 zvolí A vertikálnu hranu, aby obišiel prekážku (obr. 10.11).



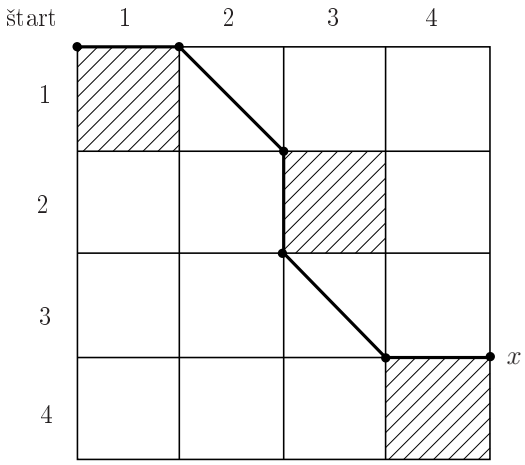
obr. 10.11

Potom nemôže protivník vytvoriť prekážku, pretože v každom stĺpci už jedna prekážka leží a urobí $A_2 = (1, 3, \mathbf{2}, \ldots)$. Teraz použije A, podľa bodu 1, pred ňou stojacu diagonálnu hranu (obr. 10.12) a dosiahne takto nový riadok a nový stĺpec. Protivník vytvorí konštrukciou

$$A_1 = (1, 2, 3, \mathbf{4}, \ldots) \text{ a } A_2 = (1, 3, 2, \mathbf{4}, \ldots)$$

novú prekážku, ktorú A obchádza horizontálne (obr. 10.12).

Ak má podnik práve 4 pracoviská, protivník je hotový s konštrukciou prípadu plánovania $x_A = (A_1, A_2)$. Na dosiahnutie cieľa (obr. 10.12) musí ešte A ísť po okraji vertikálnou hranou. Spolu použil algoritmus A na ceste zo štartu do cieľa šesť hrán a len dve z toho boli diagonálne. Algoritmus



obr. 10.12

A sa začal horizontálnou hranou a nanajvýš každá druhá hrana bola diagonálna.

□

Úloha 10.9 Predpokladajme, že podnik má 7 pracovísk. Prevezmite úlohu protivníka v pokračovaní príkladu 10.2 z vrcholu x na obrázku 10.12 a doplňte pre tento prípad konštrukciu ťažkého prípadu x_A.

Úloha 10.10 Ako vyzerá optimálne riešenie pre prípad úlohy $A_1 = (1, 2, 3, 4)$ a $A_2 = (1, 3, 2, 4)$ na obrázku 10.12?

Úloha 10.11 Zmeňme stratégiu A z príkladu 10.2 tým, že v bode 2 budeme vyžadovať, aby v prípade prekážky online algoritmus použil vždy horizontálnu hranu. Hrajte úlohu protivníka pre túto stratégiu A' a skonštruujte týmto spôsobom ťažký prípad pre A' a pre $n = 5$.

Úloha 10.12 Uvažujme stratégiu B, ktorá nezávisle od prípadu úlohy použije najprv 3 vertikálne hrany (t.j. nechá čakať objednávka A_1 tri časové jednotky) a potom pracuje ako A. Hrajte úlohu protivníka pre B a zostrojte ťažký prípad pre B pre $n = 7$.

Ukázali sme, že pre každý online algoritmus A sa dá skonštruovať ťažký prípad úlohy x_A, ktorý A nevie vyriešiť rýchlejšie ako za

$$m + m/4$$

časových jednotiek.

Je to veľa alebo málo? Aby sme túto otázku mohli zodpovedať, musíme najprv skúmať, ako dobré sú riešenia, ktoré sa dajú nájsť, ak je budúcnosť (celé objednávky) od počiatku známa. Najprv objasníme, že každý prípad plánovania pre dve objednávky na m pracoviskách je spracovateľný v čase

$$m + \sqrt{m}.$$

Dôsledkom toho je, že pre každý online algoritmus A platí:

$$\text{Konk}_A(I) \geq \frac{m + 0,25 \cdot m}{m + \sqrt{m}}.$$

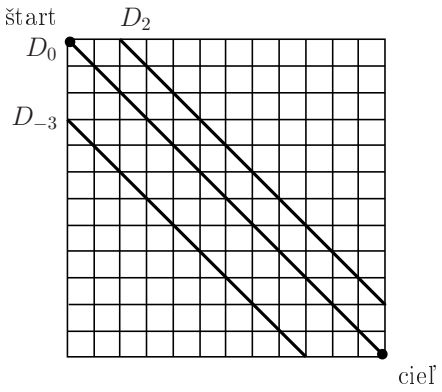
Pre veľké m to znamená, že online vypočítané riešenia môžu byť o takmer 25% horšie ako optimálne.

Aby sme ukázali, že každý prípad úlohy možno riešiť v $m + \sqrt{m}$ časových jednotkách, uvedieme pre väčšinu čitateľov úplne nový druh argumentácie. Pre každý prípad problému budeme brať do úvahy viacero algoritmov a ukážeme, že riešenia vyprodukované týmito algoritmami vyžadujú v priemere $m + \sqrt{m}$ časových jednotiek. Ak riešenia potrebujú v priemere $m + \sqrt{m}$ časových jednotiek, tak medzi nimi musí byť jedno, ktoré potrebuje nanajvýš $m + \sqrt{m}$ časových jednotiek. Odôvodnenie tejto úvahy je jednoduché. Ak by všetky riešenia potrebovali viac ako $m + \sqrt{m}$ časových jednotiek, tak priemer počtu časových jednotiek týchto riešení nemôže byť $m + \sqrt{m}$.

Pre zjednodušenie uvažujme, že $m = k^2$ pre nejaké prirodzené často k, a teda $\sqrt{m} = k$ je celé číslo. Budeme uvažovať $2k+1$ diagonálnych stratégii. Hlavnú diagonálu celej tabuľky zo štartu do cieľa označíme D_0. Diagonálu v tabuľke, ktorá leží i políčok nad D_0 označíme D_i. Analogicky označíme D_{-j} diagonálu, ktorá leží j políčok pod D_0. Na obr. 10.13 vidíme tri hrubé čiary, predstavujúce diagonály D_0, D_2 a D_{-3}.

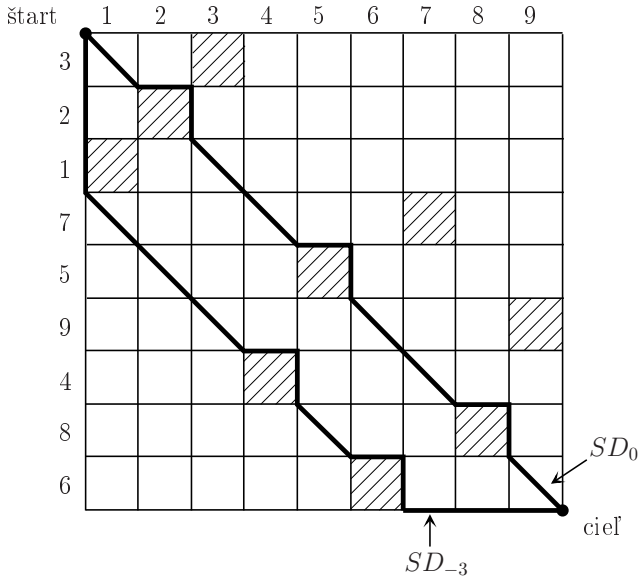
Každej diagonále D_l môžeme priradiť jednu stratégiu SD_l, ktorá sa usiluje prejsť všetkými vrcholmi diagonály D_l. Pre každé $i \geq 0$ urobí SD_i najprv i horizontálnych krokov a dosiahne tak najvyšší vrchol diagonály D_i. Ak narazí na prekážku, tak SD_i obchádza túto prekážku najprv horizontálnou a potom vertikálnou hranou (obr. 10.14) s cieľom vrátiť sa po obchádzke na D_i. Ak SD_i dosiahne najspodnejší vrchol diagonály D_i, tak na dosiahnutie cieľa SD_i použije vertikálne hrany na okraji.

Analogicky sa začína stratégia D_{-i} najprv s i vertikálnymi hranami a dosiahne tak najvyšší vrchol diagonály D_{-i}. Potom pokračuje, podobne ako SD_i, po diagonálnych hranách diagonály D_{-i} a obchádza prekážky



obr. 10.13

v dvoch krokoch s nárastom na D_{-i} (obr. 10.14). Ak SD_{-i} dosiahne najnižší vrchol diagonály D_{-i}, na dosiahnutie cieľa použije SD_{-i} horizontálne hrany.



obr. 10.14

Na obrázku 10.14 vidíme riešenie stratégií SD_0 a SD_{-3} pre prípad úlohy

$$A_1 = (1, 2, 3, 4, 5, 6, 7, 8, 9) \text{ a } A_2 = (3, 2, 1, 7, 5, 9, 4, 8, 6).$$

Úloha 10.13 Vypočítajte riešenia, ktoré dostaneme použitím stratégií SD_1, SD_2 a SD_{-2} pre prípad úlohy z obr. 10.14.

Úloha 10.14 Predstavme si prípad úlohy $A_1 = (9, 8, 7, 6, 5, 4, 3, 2, 1)$ a $A_2 = (9, 7, 8, 4, 5, 6, 2, 3, 1)$. Načrtnite riešenia, ktoré dostaneme použitím stratégií SD_3, SD_0, SD_{-1} a SD_{-2}.

Predpokladajme, že pre každý prípad plánovania s m pracoviskami použijeme všetkých $2k+1 = 2\sqrt{m}+1$ diagonálnych stratégií a získame takto $2k+1$ rôznych riešení. Vypočítame priemernú kvalitu týchto riešení tak, že spočítame najprv súčet ich časov spracovania, a potom ho vydelíme $2k+1$.

Čas výpočtu riešenia SD_i a SD_{-i} je presne

$$m + i + \text{počet prekážok na } D_i(D_{-1}),$$

pretože:

(i) SD_i (SD_{-i}) používa presne i vertikálnych a i horizontálnych hrán, aby dosiahla zo štartu diagonálu D_i (D_{-i}) a z konca diagonály prišla do cieľa. Preto majú obe objednávky zdržanie presne i časových jednotiek.

(ii) Každú prekážku obídeme v riešení jednou horizontálnou a jednou verikálnou hranou. Preto znamená každá prekážka oneskorenie o jednu časovú jednotku.

Označme pomocou SUM súčet všetkých oneskorení všetkých $2k+1$ riešení. Čitateľ, ktorý nemá vzťah k matematike, môže nasledujúci výpočet preskočiť.

$$\begin{aligned}
\text{SUM} &= \sum_{i=-k}^{k} (|i| + \text{počet prekážok na } D_i) \\
&= \sum_{i=-k}^{k} |i| + \sum_{i=-k}^{k} \text{počet prekážok na } D_i \\
&\leq 2 \cdot \sum_{i=1}^{k} i + m
\end{aligned}$$

{Toto bol najdôležitejší krok v našej úvahe. Keďže všetkých prekážok je m, nemôže byť súčet všetkých prekážok na $2k+1$ vybraných diagonálach väčší ako m. Preto je hodnota druhej sumy zhora ohraničená číslom m.}

$$= 2 \cdot \tfrac{k \cdot (k+1)}{2} + m$$

{Už malý Gauss ukázal, že súčet prvých k kladných celých čísiel je presne $k \cdot (k+1)/2$.}

$$= k \cdot (k+1) + m = k^2 + k + m = (\sqrt{m})^2 + \sqrt{m} + m$$
$$= 2m + \sqrt{m}.$$

Vydeľme súčet všetkých oneskorení počtom riešení. Dostávame priemerné oneskorenie:

$$\frac{2m + \sqrt{m}}{2 \cdot \sqrt{m} + 1} = \sqrt{m} = k.$$

Z tohto dôvodu existuje aspoň jedno riešenie, ktoré sa oneskoruje nanajvýš o k časových jednotiek[8]. Teda existuje riešenie, ktoré vybaví všetky objednávky prípadu úlohy za čas nanajvýš

$$m + \sqrt{m}$$

časových jednotiek.

Úloha 10.15 (tvrdý oriešok) Pre naše úvahy sme použili $2\sqrt{m}+1$ diagonálnych stratégií. Aký by bol výsledok nášho usília, keby sme počítali s $4\sqrt{m}+1$ alebo $\sqrt{m}+1$ diagonálnymi stratégiami?

Úloha 10.16 Vypočítajte priemerné zdržanie pre 7 stratégií od SD_3 po SD_{-3} pre plánovanie na obr. 10.14.

Dokázali sme, že pre každý prípad plánovania dvoch objednávok pre m pracovísk je možné objednávky spracovať s oneskorením najviac $\sqrt{m}$. Žiaľ, online algoritmy nie sú schopné zabrániť oneskoreniu $m/4 = 0.25m$. Pre veľké hodnoty m je $m/4$ podstatne viac ako $\sqrt{m}$. Náš výsledok získame vďaka jednoduchej kombinatorickej úvahe, ktorá je napriek svojej jednoduchosti mocným a často používaným nástrojom matematických zdôvodnení. Toto kombinatorické tvrdenie môžeme sformulovať takto:

> *Keď máme m objektov a každému priradíme nejakú číselnú hodnotu a priemerná hodnota objektov je d, potom medzi týmito objektmi existuje jeden s číselnou hodnotou najviac d a jeden s hodnotou aspoň d.*

Nadišiel čas prekabátiť prešibaného protivníka. Zoberieme si na pomoc náhodu a pre úlohu plánovania navrhneme randomizovaný online algoritmus. Všimnite si, že všetky stratégie SD_i sú online algoritmy. Každá diagonálna stratégia sa drží vrcholov svojej diagonály a obchádza prekážky rovnako bez ohľadu na to, ako sú rozmiestnené. Na svoje rozhodnutia

[8] Ak by všetky riešenia mali oneskorenie aspoň $k+1$ časových jednotiek, potom by priemerné oneskorenie muselo byť viac ako k.

nepotrebuje informácie o prípade úlohy okrem aktuálnych požiadaviek objednávok A_1 a A_2. Ako nám to pomôže? Vytvoríme randomizovaný online algoritmus D, ktorý:

> *Pre každý prípad úlohy plánovania s m pracoviskami zvolí náhodne jednu z $2\sqrt{m}+1$ stratégií SD_i a použije ju na určenie riešenia.*

Keďže priemerné oneskorenie všetkých $2\sqrt{m}+1$ stratégií je $\sqrt{m}+\frac{1}{2}$, očakávame od algoritmu D relatívne dobré riešenia. Samozrejme, nemôžeme vylúčiť, že pre daný neznámy prípad úlohy si D náhodne zvolí nevhodnú stratégiu. Napríklad, keď si randomizovaný online algoritmus D zvolí pre preňho neznámy prípad úlohy

$$A_1 = (1, 2, 3, \ldots, m) = A_2(1, 2, 3, \ldots, m)$$

diagonálnu stratégiu SD_0, leží všetkých m prekážok práve na diagonále D_0 a oneskorenie je najhoršie možné, t.j. m. To sa ale stane len s pravdepodobnosťou

$$\frac{1}{2\sqrt{m}+1}.$$

Vzhľadom na to, že na ostatných diagonálach neležia nijaké prekážky, sú všetky ostatné prípady priaznivé. V týchto prípadoch má SD_i oneskorenie presne i, pre všetky i.

Úloha 10.17 Nech $m = 9$. Nájdite taký prípad úlohy, že všetky políčka diagonály D_3 obsahujú prekážky. Určte jednotlivé oneskorenia všetkých diagonálnych stratégií.

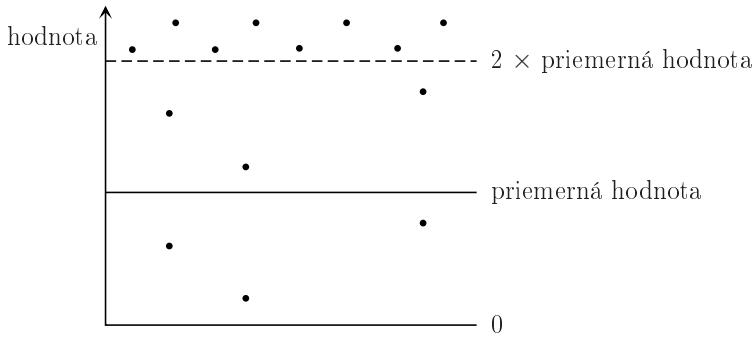
Ako sa môžeme pomocou matematických argumentov presvedčiť, že randomizovaný online algoritmus D je dobrý pre prax? Nemôže sa stať, že príliš často (s vysokou pravdepodobnosťou) dostaneme nie príliš dobré riešenia? Aby sme ukázali kvalitu algoritmu D, použijeme elegantnú kombinatorickú úvahu.

Majme n objektov, ktoré sú označené kladnými celými číslami. V našom prípade sú objekty jednotlivé riešenia vytvorené diagonálnymi stratégiami a číselné označenia sú ich oneskorenia. Tvrdíme, že:

> *Aspoň polovica objektov má hodnotu menšiu alebo rovnú dvojnásobku priemernej hodnoty.*

Ako to zdôvodníme? Nech je d priemerná hodnota. Ak viac ako polovica objektov má hodnotu väčšiu ako $2d$, tak priemer nemôže byť rovný d, musí

byť väčší ako d aj keby všetky ostatné objekty mali priradenú hodnotu nula (obr. 10.15).



obr. 10.15

Dá sa to ukázať aj výpočtom. Nech g je počet objektov s hodnotou väčšou ako $2d$ a h je počet objektov s hodnotou najviac $2d$. Je zrejmé, že $g + h$ je počet všetkých objektov. Potom je súčet hodnôt všetkých objektov dohromady viac ako

$$g \cdot 2d + h \cdot 0 = 2dg.$$

Priemerná hodnota d je potom aspoň

$$\frac{2dg}{g+h}.$$

Teda platí $d \geq 2dg/(g+h)$ a dostávame, že

$$\begin{array}{rcll} d \cdot (g+h) & \geq & 2dg & | \cdot \frac{1}{d} \\ g+h & \geq & 2g & | - g \\ h & \geq & g. & \end{array}$$

Výsledok hovorí, že počet objektov g s hodnotami väčšími ako $2d$ (dvojnásobok priemeru) nie je väčší ako počet objektov h s hodnotami najviac ako dvojnásobok priemeru. Dôsledkom nerovnosti $h \geq g$ je, že pre algoritmus D máme zaručené, že s pravdepodobnosťou najmenej $1/2$ zvolí stratégiu, ktorej riešenie má oneskorenie najviac

$$2 \cdot d = 2 \cdot \sqrt{m}.$$

Ak to niekomu ešte nestačí, môže použitú kombinatorickú úvahu zovšeobecniť:

Počet objektov s hodnotami väčšími ako c-násobok priemeru je najviac c-tina (t.j. n/c) všetkých objektov.

Ak zvolíme $c = 2$, dostaneme pôvodné tvrdenie.

Úloha 10.18 (tvrdý oriešok) Zdôvodnite platnosť zovšobecneného kombinatorického tvrdenia.

Na základe tohto kombinatorického tvrdenia môžeme povedať, že s pravdepodobnosťou aspoň 3/4 nájdeme riešenie, ktoré v najhoršom prípade spôsobí oneskorenie $4d$. Tento výsledok dosiahneme voľbou $c = 4$ v zovšeobecnenom kombinatorickom tvrdení.

Úloha 10.19 Akú záruku môžeme dať pre D na najväčšie možné oneskorenie s pravdepodobnosťou aspoň 9/10? Aká je horná hranica na veľkosť oneskorenia 7/8 všetkých diagonálnych stratégií s minimálnymi oneskoreniami?

10.4 Zhrnutie alebo ako sme prekabátili protivníka

Pre množstvo procesov prebiehajúcich v praxi sú online problémy každodenným chlebom. Obzvlášť v oblasti služieb všetkého druhu nie je možné spoľahlivo predpovedať rozsah a štruktúru požiadaviek zákazníkov. Servisné strediská sa musia rozhodovať a realizovať svoje rozhodnutia bez toho, aby vedeli, aké požiadavky ich v blízkej budúcnosti očakávajú. V mnohých situáciách majú riadiace centrá služieb chabé vyhliadky aplikovať nejakú rozhodovaciu stratégiu, ktorá by za nijakých okolností neviedla k neuspokojivému vývoju.

Úlohou algoritmikov je skúmať, pre ktorý druh online úloh máme možnosť navrhnúť online algoritmy garantujúce také dobré riešenia, ako keby sme poznali budúcnosť (všetky budúce požiadavky), a pre ktoré úlohy to nie je možné. Aby sme vedeli analyzovať rôzne situácie, hráme hru s dvomi hráčmi, navrhovateľom algoritmu a jeho protivníkom, navrhovateľom ťažkých prípadov úloh. Navrhovateľ vyvinie online algoritmus a jeho protivník sa snaží nájsť prípady úlohy, na ktorých tento algoritmus zlyhá. V hre má protivník značnú výhodu, pretože pozná navrhovaný algoritmus a môže zostaviť budúcnosť tak, aby bola pre algoritmus nepriaznivá. Preto je aj obťažné nájsť skutočne dobré online algoritmy.

No riadenie algoritmu pomocou náhodných procesov môže byť aj v prípade online algoritmov riešením zdanlivo bezvýchodiskových situácií. Z po-

hľadu hry stráca protivník v hre proti randomizovanému algoritmu svoju hlavnú výhodu. Protivník pozná síce do najmenších detailov randomizovaný algoritmus, ale nemôže predpovedať jeho rozhodnutia v konkrétnych situáciách, pretože tieto rozhodnutia sú náhodné a uskutočnia sa až v priebehu aplikovania randomizovaného algoritmu. V našom prípade s $2\sqrt{m}+1$ stratégiami musí protivník naraz hrať proti všetkým $2\sqrt{m}+1$ stratégiám, pretože netuší, ktorá z nich bude náhodne zvolená. A protivník nemá šancu nájsť prípad úlohy, ktorý by bol ťažký čo len pre zlomok týchto stratégií, pretože prípady, ktoré by boli obťažné pre viacej stratégií jednoducho neexistujú. To trocha pripomína futbalový zápas medzi dvoma mužstvami A a B. Ak hrá družstvo A podľa nejakej pevnej stratégie nezávisle od vývoja zápasu a tréner mužstva B ju odhalí, môže svojich hráčov doviesť vhodnou taktikou k úspechu. Ak ale mužstvo A hrá veľmi flexibilne a s veľkou mierou nevypočítateľnej hernej improvizácie (správa sa takmer „náhodne“), tak je veľmi ťažké, aby tréner mužstva B proti nemu našiel nejakú vhodnú stratégiu.

Z pohľadu návrhu randomizovaného algoritmu možno mať často úspech, ak máme k dispozícii skupinu deterministických algoritmov, ktoré pre väčšinu prípadov úlohy nájdu dobré riešenia a každý z nich zlyhá len pre malý počet prípadov úlohy. V takomto prípade stačí jednoducho zvoliť jeden z týchto algoritmov a použiť ho na riešenie daného prípadu úlohy. Keď každý deterministický algoritmus z tejto skupiny algoritmov zlyhá na rôznych prípadoch úlohy, takto navrhnutý randomizovaný algoritmus nájde s vysokou pravdepodobnosťou veľmi dobré riešenie.

Návody na riešenie vybraných úloh

Úloha 10.2 Pre každý prípad úlohy I je

$$\text{Cena}(\text{Riešenie}_A(I)) - \text{Opt}_U(I)$$

absolútny rozdiel medzi optimálnou cenou a cenou riešenia vypočítaného algoritmom A. Hodnota

$$\frac{\text{Cena}(\text{Riešenie}_A(I)) - \text{Opt}_U(I)}{\text{Opt}_U(I)} \cdot 100$$

hovorí, o koľko percent je vypočítané riešenie horšie ako optimálne.

Úloha 10.4 V Cache máme stránky $1, 3, 5$ a 7. V prípade potreby odstránime vždy stránku s najmenším číslom. Pre túto stratégiu protivník skonštruuje nasledujúci prípad úlohy:

$$2, 1, 2, 1, 2, 1, 2, 1, 2, 1.$$

Online stratégia odstránenia stránky s najmenším číslom skončí pre tento prípad s riešením:

$$1 \leftrightarrow 2 \quad , \quad 2 \leftrightarrow 1 \quad , \quad 1 \leftrightarrow 2 \quad , \quad 2 \leftrightarrow 1 \quad , \quad 1 \leftrightarrow 2$$

$$2 \leftrightarrow 1 \quad , \quad 1 \leftrightarrow 2 \quad , \quad 2 \leftrightarrow 1 \quad , \quad 1 \leftrightarrow 2 \quad , \quad 2 \leftrightarrow 1$$

Zrejme je

$$5 \leftrightarrow 1 \quad , \quad \bullet \quad , \quad \bullet \quad , \quad \bullet \quad , \quad \bullet \quad , \quad \bullet \quad , \quad \bullet \quad , \quad \bullet \quad , \quad \bullet \quad , \quad \bullet$$

optimálne riešenie, ktoré je desaťkrát lepšie ako riešenie uvažovanej stratégie.

Úloha 10.6 Pre prípad úlohy $A_1 = (1, 2, 3, 4)$ a $A_2 = (3, 2, 1, 4)$ poskytuje nasledujúce rozdelenie práce:

časové jednotky	1	2	3	4	5
A_1	S_1	S_2	S_3	S_4	
A_2		S_3	S_2	S_1	S_4

optimálne riešenie.

Úloha 10.16 Všetkých 9 prekážok leží na siedmich uvažovaných diagonálach (obr. 10.14). Na D_0 ležia tri prekážky, takže s SD_0 dosiahneme oneskorenie $d_0 = 3$. Na D_1 a D_{-1} neležia nijaké prekážky, a teda celkové oneskorenie stratégií SD_1 a SD_{-1} je 1 ($d_1 = d_{-1} = 1$). Oneskorenie je spôsobené dosiahnutím a opustením diagonály. Na D_2 a na D_{-2} leží po jednej prekážke, teda oneskorenie oboch prípadov je $d_2 = d_{-2} = 3$. Na D_3 a D_{-3} leží po dvoch prekážkach, takže $d_3 = d_{-3} = 3 + 2 = 5$. Priemerné oneskorenie týchto sedem stratégií je

$$\frac{d_0 + d_1 + d_{-1} + d_2 + d_{-2} + d_3 + d_{-3}}{7} = \frac{3 + 1 + 1 + 3 + 3 + 5 + 5}{7} = 3.$$

Úloha 10.17 Prípad problému $A_1 = (1, 2, 3, 4, 5, 6, 7, 8, 9)$ a $A_2 = (4, 5, 6, 7, 8, 9, 1, 2, 3)$ sa vyznačuje tým, že všetkých 6 políčok diagonály D_3 obsahuje prekážky.

Čo bolo pre nás typické?
Nemali sme strach
povedať mladým ľuďom,
že my sme sami hlúpi...

Niels Bohr

Kapitola 11

Algoritmická optimalizácia vo fyzike alebo ako by mohla liečiť homeopatia?

11.1 O dôveryhodnosti vedeckých teórií

Táto posledná kapitola sa podstatne odlišuje od predchádzajúcich. Všetko, čo sme doteraz z vedy ukázali, považujeme za fakty. Na jednej strane sme sa naučili, že základné stavebné kamene vedy sú do istej miery vecou viery a dôvery. Na druhej strane sme si tiež uvedomili, že vedci vybudovali budovu vedy veľmi starostlivo a že všetky tehličky, ktoré boli položené na základné kamene, boli veľmi pozorne teoreticky ako i experimentálne preskúmané. Matematika, informatika a prírodné vedy dosahujú stále vyššiu zrelosť a tým aj vyššiu stabilitu. Vďaka tomu sú teórie a z nich odvodené predpovede čoraz spoľahlivejšie. No bolo by mylné sa nazdávať, že celá veda je takáto a že takou aj vždy bola. Na relatívne pevnej pôde stojíme len tam, kde máme možnosť robiť spoľahlivé experimenty a merania a kde môžeme opísať skutočnosť jazykom matematiky. Ale ako je to s pedagogikou, didaktikou, sociológiou, ekonómiou a inými vedeckými disciplínami, v ktorých sa vedci nevedia dohodnúť na obraze reality a vytvárajú navzájom si odporujúce teórie? Zažili sme už veľakrát, že tieto disciplíny od základov menili svoje predstavy. Robia to aj naďalej

a môžeme očakávať, že mnohé dnešné „pravdy“ už zajtra platiť nebudú. Neslobodno to chápať ako kritiku týchto vied. Chceme len povedať, že materiál týchto vedeckých disciplín je taký zložitý a základné pojmy nie sú dostatočne precízne a v tomto štádiu ich vývoja to pravdepodobne ani ináč nemôže byť. Pred storočiami nevyzerala fyzika a čiastočne ani matematika oveľa lepšie. Musíme sa zmieriť s tým, že rôzne vedecké disciplíny sa nachádzajú v rôznych štádiách vývoja. Tam, kde chýbajú pevne definované pojmy a kde nevieme z pozorovaného merať a posudzovať to, čo by sme chceli, ostáva príliš veľa miesta na úvahy, ktoré sa javia z pohľadu matematiky a fyziky ako špekulácie. Ale tieto špekulácie sú potrebné, pretože hľadanie pravdy vo vede vyžaduje tisíce omylov a neúspešných pokusov. Exaktné vedy ruší skôr skutočnosť, že humanitné disciplíny nevyužívajú v dostatočnej miere existujúce možnosti a nástroje, ktoré by mohli pomôcť spoľahlivejšie overovať vyslovené hypotézy a tvrdenia[1]. Skúsme byť konkrétni a pozrime sa na didaktiku informatiky. Predstavme si, že sa chceme presvedčiť, že výučba nejakej témy prostredníctvom istej didaktickej metódy (napr. objektovo orientované programovanie priamo za obrazovkou) je vhodnejšia ako výučba inej príbuznej témy inou metódou (napr. štruktúrované programovanie na papieri s ceruzkou a gumou). Ako sa ale dá posúdiť, čo je vhodnejšie? A čo vôbec znamená vhodnejšie? Už len meranie získaných vedomostí a dosiahnutej kompetencie je obťažné, ak má výsledok nášho merania vyústiť do spoľahlivej výpovede. A už vôbec nie sme schopní merať prínos k vývoju spôsobu uvažovania žiakov. Predpokladajme ale, že sme schopní formulovať úlohy a systém hodnotenia riešení, ktoré nám umožnia aspoň do istej miery posudzovať niektoré dôležité aspekty vyučovacieho procesu. V tejto situácii sa od nás požaduje príprava experimentu. V ideálnom prípade potrebujeme rôzne triedy, ktorých žiaci majú približne rovnaké schopnosti a vedomosti. Naviac potrebujeme učiteľov, ktorí vedia približne rovnako dobre porovnávané témy vyučovať. Už splnenie týchto predpokladov nevieme spoľahlivo overiť. Napriek tomu začneme s pokusom, pričom sa musíme zmieriť s tým, že nemáme možnosť zohľadniť dôležité vplyvy, akými sú atmosféra v triede a pomoc rodičov pri učení sa doma. Nepochybne musíme za daných okolností výsledok nášho úsilia interpretovať veľmi opatrne. A zásadne sme povinní upozorniť používateľov našich výsledkov na ich relativitu. Napriek tomu, že všetko je také relatívne a neisté, neostáva nám nič iné, pretože za súčasných znalostí v súčasnom stave vývoja didaktiky spoľahlivejšie výskumné metódy neexistujú. Vďaka rozpoznaným omylom a chybným záverom sa učíme „merať“ stále presnejšie, dávať si realistickejšie ciele, experimentovať korektnejšie, obmedzovať pritom ne-

[1] napríklad matematickými metódami a modelmi

želaný vplyv a naše pozorovania relativizovať. Takto pracuje aspoň časť etablovaných didaktikov v matematike alebo fyzike. Bolo by nesprávne, keby sme sa kvôli neistotám v meraní vôbec nepokúsili experimentovať a keby sme tvorili teórie len na základe takzvaných „rozumných“ úvah. Takto by sme dochádzali k skutočným špekuláciám a sporným záverom, ktoré sú, žiaľ, v dnešnej didaktike informatiky skôr typickým ako mimoriadnym zjavom[2].

Niekoho prekvapí, že používanie formálneho jazyka matematiky môže byť príjemnejšie a spoľahlivejšie napriek tomu, že zvládnuť tento jazyk je veľmi náročné. Pevná pôda exaktných vied, akými sú matematika, fyzika, chémia alebo informatika, dáva pocit istoty, ktorý sa nedá dosiahnuť na pôde vedných disciplín, v ktorých človek musí mať obrovské skúsenosti, aby dokázal rozlíšiť špekulatívne od zmysluplného. Táto skutočnosť ovplyvnila vo väčšine krajín výber vyučovacích predmetov na stredných školách. Napriek tomu hodláme v tejto kapitole opustiť pevnú pôdu exaktných vied a riskovať výlet do sveta nejasnej terminológie a matematicky zatiaľ nepostihnuteľnej skutočnosti. Počas tohto výletu sa ale budeme stále usilovať o to, aby čitateľ vedel rozlišovať medzi predpokladmi, hypotézami a ich dôsledkami a bol si stále vedomý relatívnosti formulovaných výpovedí.

Naše úvahy začneme na pevnej pôde fyziky. Budeme sa zaoberať optimalizáciou kryštálovej štruktúry kovov a naším cieľom bude matematicky modelovať proces optimalizácie pomocou jedného algoritmu. Pritom objavíme, že tento algoritmus sa dá úspešne využiť na riešenie algoritmických optimalizačných úloh. Prekvapí nás však, že nie sme schopní nájsť priamu súvislosť medzi použitým fyzikálnym princípom a štruktúrou optimalizačných úloh. Tu opustíme teóriu a vydáme sa cestou experimentov a skúseností, ktoré môžeme len čiastočne vysvetliť prostredníctvom exaktných prostriedkov. Nato definitívne opustíme pevnú pôdu a konfrontujeme sa s otázkou, či študovaný fyzikálno-algoritmický princíp optimalizácie nie je využiteľný aj na modelovanie procesu liečenia živých bytostí. Tieto úvahy nás povedú k pokusu definovať zdravie a chorobu v terminológii fyziky a algoritmiky. Takéto definície nám potom umožnia nahliadnuť na liečenie ako na proces algoritmického spracovania informácie. V konečnom dôsledku naše úsilie vyústi vo formulácii predstáv o možnom pôsobení alternatívnych liečebných metód ako je napríklad homeopatia.

[2]Nie je to len dôsledkom toho, že didaktika informatiky je veľmi mladá a ešte si nevytvorila rozumné výskumné štandardy. Kvôli nedostatku informatikov sú aj v didaktike informatiky činní ľudia, ktorí prišli z iných oblastí, alebo sú to čistí používatelia informačných technológií. Títo sa nikdy nestretli s výskumom informatických základov, a preto si vôbec neuvedomujú všeobecnovzdelávacie hodnoty informatiky.

11.2 Algoritmický model optimalizácie kryštalickej štruktúry kovov

Na začiatok prinášame prvú prekvapujúcu správu:

> Kov „vie“, čím je a po celý čas svojej existencie sa usiluje byť tým, čím je.

Ako správne pochopiť túto vetu o takzvanej mŕtvej hmote? Kovy sa „usilujú“ po celý čas svojej existencie dosiahnuť a udržať svoju optimálnu kryštalickú štruktúru, ktorou sa vyznačujú a odlišuje ich od ostatných prvkov. V tomto zmysle sa každý kov nepretržite „usiluje“ sám seba optimalizovať. Optimalizovať znamená, že sa energia rovnomerne rozdelí na väzby medzi jednotlivými časticami, a takto sa minimalizuje takzvaná **voľná energia**. Napriek neustálemu úsiliu môže dôjsť v dôsledku vonkajších vplyvov a záťaže k takzvanej „únave“ kovu. Pre túto únavu sa môžu väzby v istých častiach postupne oslabiť, čo môže v najhoršom prípade vyústiť do poškodenia materiálu, napr. do zlomu. Ako môžeme kovu pomôcť zlepšiť jeho stav? Na základe zákonov termodynamiky potrebuje kovový materiál „horúci kúpeľ“. Optimalizačný proces stavu kovu prostredníctvom horúceho kúpeľa pozostáva z dvoch fáz:

- **Prvá fáza**

  Pomocou horúceho kúpeľa dodáme kovu zvonka energiu. Toto veľké množstvo voľnej energie oslabí väzby a privedie kov do stavu, ktorý je do značnej miery chaotický. Z pohľadu množstva voľnej energie je tento stav jednoznačne zhoršením predchádzajúceho stavu.

- **Druhá fáza**

  Kov necháme pomaly vychladnúť. V priebehu ochladzovania preváži úsilie kovu o dosiahnutie svojho optimálneho stavu. Pri dostatočne pomalom ochladzovaní dosiahne kov vďaka „vlastnému úsiliu“ stav, ktorý je blízky optimálnemu.

Fyzici [MRR$^+$53] objavili nasledujúci algoritmus, ktorý dnes nazývame **Metropolisov algoritmus**. Tento algoritmus umožňuje verne simulovať hore opísaný optimalizačný proces. Na opísanie Metropolisovho algoritmu používame označenie $\boldsymbol{E(s)}$ pre voľnú energiu stavu s optimalizovaného kovu. Symbolom C_B označujeme Boltzmannovu konštantu.

Metropolisov algoritmus

- **Vstup**

  Stav s kovu[3] s energiou $E(s)$.

- **Prvá etapa**

  Urči počiatočnú teplotu T (v Kelvinoch) horúceho kúpeľa.

- **Druhá etapa**

  Pomocou náhodnej malej zmeny (napr. zmeny polohy jednej častice) vygeneruj z aktuálneho stavu s nejaký stav q.

  Keď $E(q) \leq E(s)$, tak q považuj za nový aktuálny stav.
  (To znamená, že ak je stav q lepší ako stav s, tak optimalizovaný kov prejde jednoznačne do nového stavu q.)

  Ak $E(q) > E(s)$, akceptuj q ako nový aktuálny stav kovu s pravdepodobnosťou

  $$\text{prob}(s \to q) = e^{-\frac{E(q)-E(s)}{C_B \cdot T}}$$

  (to znamená, ostaň v pôvodnom stave s pravdepodobnosťou $1 - \text{prob}(s \to q)$).

- **Tretia etapa**

  Vhodne redukuj T (zníž teplotu horúceho kúpeľa).

  Ak T (v Kelvinoch) nie je blízko k 0, pokračuj druhou etapou.

  Ak je T (v Kelvinoch) blízko k 0, ukonči prácu algoritmu a daj na výstup stav s.

Úspešná simulácia optimalizácie pomocou Metropolisovho algoritmu je založená na zákonoch termodynamiky, ktorým sa nechceme bližšie venovať. Pre jeho pochopenie je podstatné nasledujúce pozorovanie. Ak náhodne vygenerujeme lepší stav, prejde kov vždy do tohto lepšieho stavu. Ak je nový stav q horší ako pôvodný stav s, tak napriek zhoršeniu celkového stavu kovu nie je vylúčené, že kov do tohto horšieho stavu prejde. Pravdepodobnosť prechodu do horšieho stavu je daná formulou založenou na fyzikálnych zákonoch. Pravdepodobnosť zhoršenia rastie s teplotou T horúceho kúpeľa a zmenšuje sa s veľkosťou zhoršenia $(E(q) - E(s))$. To znamená, že priebehom času (chladnutím) pravdepodobnosť zhoršenia klesá, zatiaľ čo na začiatku procesu optimalizácie je vysoká.

[3] Opísaný stavmi a pozíciami všetkých častíc.

Na základe fyzikálnych zákonov môžeme matematicky odvodiť nasledovné skutočnosti:

(i) Počas procesu ochladzovania sa voľná energia minimalizuje a pri dostatočne dlhom čase dosahuje[4] kov stavy blízke optimu s takmer perfektnou kryštalickou štruktúrou.

(ii) Pozitívna pravdepodobnosť zhoršenia súčasného stavu (najmä na začiatku procesu optimalizácie) je nevyhnutnou súčasťou optimalizácie. Bez nej nemožno úspešne optimalizovať.

Prvé poučenie z Metropolisovho algoritmu je, že zlepšenie nie je možné vždy dosahovať priamočiaro. Aby sme dosiahli podstatné zlepšenia, musíme byť ochotní akceptovať počiatočné zhoršenia. Táto skutočnosť nás nesmie prekvapiť, pretože príroda je taká vo svojej podstate. Keď renovujeme dom, musíme tiež najprv dosť vecí a častí poškodiť alebo zničiť. Až potom môžeme začať vylepšovať. A ak chceme zmeniť zakorenený politický systém, môže jeho transformácia vyzerať oveľa dramatickejšie. Revolúcie v dejinách ľudstva zaznamenali na začiatku skoro vždy katastrofálne zhoršenia, a to platí dokonca aj v prípadoch, keď uskutočňované zmeny boli nevyhnutnosťou na ceste k novej, lepšej spoločnosti. Väčšina ľudí nie je schopná akceptovať, že začiatok pozitívneho vývoja môže byť spojený s nemalými zhoršeniami. Nevhodné daňové systémy, nefunkčné alebo nespoľahlivé zdravotníctvo či neuspokojivé školstvo sa nedajú v mnohých krajinách vylepšiť bez podstatných zásahov spojených aj s istými stratami.

11.3 Optimalizácia v informatike podľa fyzikálnych zákonov

Optimalizácia pomocou Metropolisovho algoritmu prebieha podľa zákonov termodynamiky. Môj pokus preniesť tento princíp do každodenného života a do vývoja spoločnosti môžete smelo nazvať špekuláciou. Nejde o nič iné, ako môj subjektívny pohľad na veci, ktorý je postavený na istých analógiách. Veľkým prekvapením je ale skutočnosť, že existujú neuveriteľne pozitívne skúsenosti s aplikovateľnosťou Metropolisovho algoritmu v oblastiach, ktoré nemajú na prvý pohľad s termodynamikou nič spoločné. V nasledujúcom chceme dokumentovať širšiu uplatniteľnosť princípu, na ktorom je Metropolisov algoritmus založený.

[4]Kov nemôže trvalo zotrvávať v jednom stave ani v prípade, že dosiahnutý stav je optimálny. Neustála zmena je nevyhnutnou charakteristikou existencie kovu.

Už sme sa naučili, čo sú to optimalizačné úlohy. Pre každý prípad problému existuje veľa prípustných riešení. Každé prípustné riešenie má svoju cenu. Našou úlohou je nájsť nejaké optimálne riešenie, čo znamená riešenie s minimálnou alebo maximálnou cenou. Mnohé z týchto úloh sú ťažko riešiteľné, pretože nepoznáme lepšiu stratégiu, ako sa pozrieť na všetky riešenia a porovnať ich ceny. Môžbyť efektívnejšia stratégia ani neexistuje. K takýmto ťažkým problémom patria veľké systémy lineárnych rovníc a nerovníc, ako ich poznáme z piatej kapitoly. Dôležité je, že nepripúšťame ľubovoľné reálne riešenia, ale len hodnoty 0 a 1, čo práve robí hľadanie optimálnych riešení systémov rovníc a nerovníc ťažkým. Rozdiel oproti kapitole 5 je ale v tom, že nemusíme nevyhnutne splniť všetky rovnice a nerovnice, ale sa usilujeme splniť ich toľko, ako sa len dá. Inými slovami, snažíme sa minimalizovať počet nesplnených rovníc a nerovníc. Pozrime sa napríklad na nasledovný systém pozostávajúci zo 7 nerovností a rovníc.

$$\begin{aligned} x_1 + x_2 + x_3 + 4x_4 &\geq 4 && (11.1)\\ x_1 - 2x_2 + 3x_3 - x_4 &\leq 2 && (11.2)\\ x_2 + x_3 &\geq 1 && (11.3)\\ x_1 + x_3 &\geq 1 && (11.4)\\ x_1 - x_4 &= 0 && (11.5)\\ 2x_1 - x_2 + 2x_3 - x_4 &\leq 2 && (11.6)\\ x_3 - x_4 &= 0 && (11.7) \end{aligned}$$

Možné riešenia sú všetky dosadenia hodnôt 0 a 1 premenným x_1, x_2, x_3, x_4. Napríklad vektor $(1, 0, 1, 0)$ označuje riešenie $x_1 = 1, x_2 = 0, x_3 = 1$ a $x_4 = 0$. Toto riešenie spĺňa (11.3), (11.4) a nespĺňa (11.1), (11.2), (11.5), (11.6) a (11.7). Takže cena riešenia je 5, pretože máme 5 nesplnených podmienok. Našou úlohou je cenu riešenia minimalizovať.

Úloha 11.1 Aká je cena riešení $(0, 0, 0, 0), (1, 1, 0, 0), (0, 1, 1, 1)$ a $(1, 1, 1, 1)$? Nájdite spomedzi 16 prípustných riešení jedno optimálne.

Iný známy príklad ťažkej optimalizačnej úlohy je problém obchodného cestujúceho, známy aj pod skratkou TSP (Traveling Salesman Problem). V ňom uvažujeme n miest, ktoré sú pospájané leteckými linkami. Z každého mesta možno priamo letieť do hociktorého iného. Každá linka medzi dvoma mestami A a B má pevnú cenu, nezáleží na tom, či letíme z A do B alebo z B do A. Riešenia sú takzvané **Hamiltonovské kružnice**. Hamiltonovská kružnica sa začína a končí v tom istom meste a každé mesto

navštívi práve raz. Cena takejto cesty (Hamiltonovskej kružnice) sú celkové náklady na letenky. Našou úlohou je minimalizovať cenu za takúto prepravu.

Tento problém musíme často riešiť v priemyselných aplikáciách. Jedna zo štandardných metód, ktorá sa aplikovala na hľadanie dobrého riešenia je takzvané **lokálne vyhľadávanie**. Táto metóda sa dá jednoducho opísať nasledujúcou schémou:

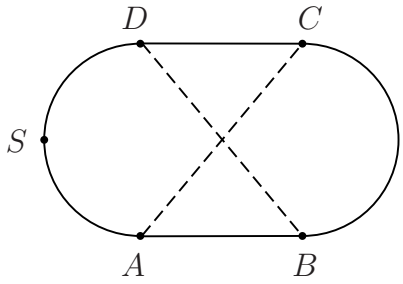
1. Pre daný prípad problému vygeneruj ľubovoľné riešenie α.
2. Prostredníctvom malých lokálnych zmien aktuálneho riešenia α hľadaj podobné riešenie, ktoré je lepšie (má menšiu cenu). Ak nájdeš nejaké lepšie riešenie β, vezmi ho ako nové aktuálne riešenie namiesto α.

   Keď pomocou lokálnych zmien nenájdeš lepšie riešenie ako α, ukonči prácu a na výstup daj riešenie α.

Aby sme túto metódu naozaj pochopili, musíme si najprv ujasniť, čo rozumieme pod lokálnymi zmenami riešenia. V prípade systému nerovníc môžeme považovať za lokálnu zmenu zmenu hodnoty jednej premennej. V tomto prípade môžeme od riešenia (0101) prejsť k riešeniu (0111) zmenou hodnoty tretej premennej z 0 na 1. Keď je nejaké riešenie dosiahnuteľné z iného riešenia vďaka lokálnej zmene, nazývame tieto riešenia **susednými**, alebo priamo **susedmi**. Teda vzhľadom na zvolenú lokálnu transformáciu sú (0110) a (0010) susedia a riešenia (0000) a (1100) nie sú susedia. Metódu lokálneho vyhľadávania môžeme v terminológii susednosti opísať takto: Pre dané riešenie hľadaj medzi jeho susedmi lepšie riešenie. Ak neexistuje lepší sused, je výstupom aktuálne riešenie.

Pre TSP môžeme definovať lokálne zmeny a tým aj susednosť nasledujúco. Z danej Hamiltonovskej kružnice odstránime dve spojenia (linky) $Lin(A, B)$ a $Lin(C, D)$ (Obr. 11.1) a nahradíme ich novými linkami $Lin(A, C)$ a $Lin(B, D)$, ako je to znázornené na obrázku 11.1. Pôvodná Hamiltonovská kružnica začínala z mesta S a navštívila medzi inými aj mestá A, B, C a D, presne v tomto poradí. Nová susedná Hamiltonovská kružnica tieto mestá navštívi v poradí A, C, B a D.

Pre mnohé prípady problému TSP poskytuje lokálne vyhľadávanie celkom dobré riešenia, ktorých cena sa veľmi nelíši od optimálnej ceny. No existujú aj prípady TSP problému, pre ktoré lokálne vyhľadávanie dáva veľmi zlé riešenie. V týchto prípadoch nemožno dosiahnuť skutočne dobré riešenie len postupnosťou neustálych zlepšení. Na dosiahnutie dobrého rie-



obr. 11.1

šenia v takejto situácii musíme v priebehu optimalizácie a najmä na jej začiatku pripustiť aj kroky, ktoré vedú k zhoršeniu riešenia. Začiatkom 80-tych rokov prišli Černý, Kirkpatrick, Gellat a Vecchi [Čer85, KGV83] s myšlienkou zlepšiť lokálne vyhľadávanie v zmysle Metropolisovho algoritmu. Výsledný algoritmus dostal názov „Simulated Annealing“ a bol úspešne použitý v tisíckach aplikácií. Transformácia Metropolisovho algoritmu na algoritmus na riešenie diskrétnych optimalizačných úloh je veľmi jednoduchá. V podstate stačí zmeniť terminológiu termodynamiky použitú v Metropolisovom algoritme na nasledujúce „synonymá“ diskrétnej algoritmickej optimalizácie:

Množina stavov systému	$\doteq$	množina prípustných riešení
energia stavu	$\doteq$	cena riešenia
optimálny stav	$\doteq$	optimálne riešenie
teplota kúpeľa	$\doteq$	parameter programu

Touto zmenou terminológie dostávame metódu simulovaného žíhania.

Simulované žíhanie

Vstup Prípad I nejakej optimalizačnej úlohy.

1. Nájdi pre I nejaké riešenie α a nastav hodnotu parametra T.
2. Vygeneruj náhodne suseda β riešenia α.
3. Ak je β lepší ako α, prevezmi β ako nové aktuálne riešenie ($\alpha \leftarrow \beta$).

   V prípade, že β nie je lepšie ako α, prevezmi β s pravdepodobnosťou danou vzorcom z Metropolisovho algoritmu.
4. Zmenši vhodne T.

Ak je T ešte dostatočne veľké, pokračuj krokom 3 s aktuálnym riešením α.

V prípade, že je T blízko 0, ukonči prácu a na výstup daj aktuálne riešenie α.

Napriek tomu, že algoritmické optimalizačné úlohy nemajú nič spoločné s termodynamikou, sme schopní matematicky dokázať, že pri vhodnej voľbe susednosti a vhodnom nastavení počiatočnej hodnoty parametra T, ako i jeho vhodnej redukcii v priebehu optimalizácie, garantuje metóda optimalizovaného žíhania pri dostatočne dlhom behu dosiahnutie optimálnych riešení. Negatívnou správou je, že čas potrebný na zaručené dosiahnutie optima je príliš dlhý na to, aby sme tak dlho mohli optimalizovať. Pravda, v praxi poskytuje táto metóda vo väčšine prípadov dobré riešenia v rozumnom čase. Prečo sú typické prípady úloh v praxi v tomto zmysle ľahké, nevieme zatiaľ teoreticky zdôvodniť. No vyplýva z toho dôležité poučenie:

Optimalizácia ako postupnosť krokov, z ktorých každý vedie k zlepšeniu, je vo väčšine prípadov nefunkčná. Keď chceme dosiahnuť podstatné zlepšenie, musíme v priebehu optimalizácie pripustiť aj kroky vedúce k dočasnému zhoršeniu.

11.4 Liečenie ako algoritmická optimalizácia zdravotného stavu

V tomto okamihu opúšťame pevnú pôdu matematických pojmov a konceptov a putujeme smerom k medicíne. Ako sme už spomínali, neexistuje fyzikálno-chemický koncept (definícia pojmu život), ktorý by nám umožňoval rozlišovať medzi živou a neživou hmotou. Nevieme dostatočne presne, čo vlastne znamená život, čo sme. O tom, aký je zmysel našej existencie, môžeme nanajvýš diskutovať. Keď ale nevieme, čo je živá hmota a živý organizmus, nevieme ani, čo je zdravie, choroba a liečenie. Môžete opäť diskutovať o tom, že isté definície týchto pojmov sme zaviedli a na tejto báze sme schopní relatívne úspešne liečiť chorých ľudí. Rovnako ako fyzici, ktorí bez toho, aby vedeli, čo je vlastne energia, urobili obrovský pokrok v skúmaní zákonitostí nášho sveta. Aj medicína dosiahla so svojimi nepresnými pojmami nemalé úspechy. Na druhej strane sme si v prvej kapitole uvedomili, že zlepšenie chápania základných pojmov, a tým aj zodpovedajúcich objektov výskumu, je kľúčové pre kvalitatívne skoky napredovania vedy.

V tejto súvislosti sa pozrime na definíciu **zdravia**, ako ju sformulovala svetová zdravotnícka organizácia v roku 1948:

> *Zdravie je stav dokonalého fyzického, psychického a sociálneho pocitu pohody, nielen neprítomnosť choroby.*

Je to nádherná a v pravom zmysle slova humánna definícia zdravia. Ako základný predpoklad zdravia nám dáva právo a nárok sa dobre cítiť a byť spokojný. Podáva súčasne aj našu skúsenosť so vznikom chorôb. Nadmerné fyzické, psychické alebo sociálne zaťaženie, strach, agresivita a nespokojnosť patria k najdôležitejším príčinám vzniku ochorení. Napriek svojím prednostiam nám táto definícia veľmi nepomôže, ak chceme študovať liečenie ako prírodný proces. Nahrádza totiž pojem zdravia synonymom „stav úplnej spokojnosti". Pomáha takto pochopiť, čo rozumieme pod zdravím, ale nedefinuje pojem zdravia. Zjednodušená definícia pojmu zdravie ako neprítomnosť choroby vyzerá na prvý pohľad azda trochu vecnejšie, ale iba za predpokladu, že by sme mali dobrú definíciu choroby, ktorá by sa obišla bez pojmu zdravia.

Stojíme pred náročnou úlohou. Bez toho, aby sme vedeli, čím sme a čo znamená zdravie, chceme študovať proces liečenia. Aby sme v našom úsilí pokročili, neostáva nám nič iné, iba uchýliť sa k niekoľkým axiómam a s ich pomocou definovať chýbajúce pojmy. Na začiatku kapitoly sme vyslovili pozorovanie, že železo „vie", čím je. Toto „vie" sme interpretovali ako neustálu snahu byť železom (samým sebou), čo znamená počas celej svoje existencie sa usilovať o dosiahnutie svojho optimálneho stavu (perfektnej kryštalickej mriežky). A keď je neživá hmota schopná neustále sa snažiť dosiahnuť svoj optimálny stav (a teda byť, čím je), niet dôvodu nepriznať túto schopnosť aj živým bytostiam. Touto úvahou sa dostávame k nasledujúcemu základnému konceptu axiomatickej povahy.

> *Človek vo svojej podstate vie, čím je a pokúša sa celý život dosiahnuť svoj optimálny stav, teda byť tým, čím je.* ***Zdravie*** *človeka je jeho optimálny stav.* ***Choroba*** *je odchýlkou od tohto stavu, ktorá môže byť rôzne veľká.* ***Liečenie*** *je postupnosť stavov človeka, ktorá sa začína v stave vzdialenom od optima a končí sa v stave blízkom optimálnemu.*

Čo táto definícia poskytuje a čo nám v nej chýba? Vytratilo sa z nej humánne posolstvo, že pre naše zdravie je dôležitý stav psychickej pohody a sociálnej spokojnosti. V našej definícii sa vôbec nepokúšame povedať, čo je zdravie nejakej bytosti. Predpokladáme len existenciu optimálneho stavu každého organizmu a veríme v existenciu neznámej fyzikálnej definí-

cie optimality. Definícia navrhuje nazerať na proces liečenia ako na proces optimalizácie. Čo nám to dáva? Predovšetkým analógiu k optimalizačným procesom vo fyzike a informatike a na ideálnej úrovni zdôraznenie úlohy vnútorných síl organizmu, ktoré sa neustále snažia optimalizovať jeho (zdravotný) stav. Takáto predstava nepretržitého úsilia nejakej bytosti o dosiahnutie svojho optimálneho stavu nie je založená len na analógii s „neživou“ hmotou kovov. Reálnosť tejto predstavy potvrdzujú tisícročiami nadobudnuté skúsenosti s liečením a nespochybniteľná aktivita nášho imunitného systému.

Ako nám pomáha táto predstava? Po prvé už pri hľadaní liečebných metód. Mnohé odvetvia medicíny sa do značnej miery vyvinuli a dosiahli nemalé úspechy na princípe lokálneho ošetrenia. Zdravotné problémy môžeme riešiť napríklad cielenými chirurgickými zákrokmi alebo pomocou chemických preparátov. I keď týmto metódam vďačíme za obrovské úspechy v liečení, nepochybujeme o tom, že každý lekársky zákrok má globálne vedľajšie účinky, ktorých dôsledky nemožno v plnej miere odhadnúť. Večné hľadanie vratkej rovnováhy medzi prevahou prospešnosti a prevahou škodlivosti zásahu dáva lekárom pri rozhodnutiach o vhodnosti konkrétnych liečebných postupov poriadne zabrať. Každá, i keď malá operácia, má svoje riziká a napríklad aj antibiotiká by sa mali užívať opatrne len vo veľmi vážnych prípadoch. Naša otázka je: Môže liečenie prebiehať aj inak? Môžete sa pýtať: Ako inak? Odpovedáme takto: Keď presne nevieme, čo v organizme spôsobíme a vyvedieme z rovnováhy aplikáciou medicínskych postupov a liekov, vynára sa otázka, či by sme nemohli riadenie liečebného postupu ponechať samotnému organizmu, lepšie povedané jeho vnútornej sile? Táto vnútorná sila vie najlepšie, čo sme a môže viesť organizmus k jeho optimálnemu stavu. Znie to síce pekne, ale zrejme to tak jednoducho nefunguje. Keby to fungovalo, boli by sme stále zdraví a nepotrebovali by sme vôbec lekársku pomoc. Naša vizionárska myšlienka je trochu iná. Hľadajme možnosti ako povzbudiť náš organizmus k intenzívnejšej snahe optimalizovať jeho súčasný stav. Železo sa môže pre silnú vonkajšiu záťaž zlomiť i napriek neustále prebiehajúcej optimalizácii. Preto používame na dosiahnutie stavu, v ktorom sa ľahšie presadí vnútorná podstata železa horúci kúpeľ. Ako môžeme oznámiť telu, že v istom stave by sa malo viac usilovať? Ak pripustíme ďalší predpoklad, že riadiaci mechanizmus organizmu je vo svojej podstate informačným procesom, pomocou prenosu informácie sa môžeme pokúsiť zúčastniť sa na jeho riadení. Samozrejme, vidíme aj podstatný rozdiel medzi kovmi a živými bytosťami. Pre nás neznámy opis optimálneho stavu zložitého organizmu je určite neporovnateľne zložitejší, ak je vôbec možná jeho

reprezentácia. Namiesto jedného parametra (voľnej energie) pri kove u živého organizmu môžeme optimalizovať množstvo rozličných parametrov, ktoré možno ani nedokážeme naraz osloviť.

Pokúsme sa zrekapitulovať našu myšlienku. Nevieme, ako vyzerá optimálny stav človeka, a teda nevieme ani, čo je zdravie. Tušíme ale, že živé bytosti v sebe nesú túto informáciu a že existujú mechanizmy, ktoré sa snažia korigovať stav organizmu k optimalite. Má teda zmysel povzbudiť (inicializovať) tieto mechanizmy k aktívnejšiemu úsiliu? Ako? Skúšaním a zbieraním skúseností, čo je tak či onak hlavná metóda medicínskeho výskumu.

Mne osobne táto myšlienka neprišla na um vďaka vedomostiam o Metropolisovom algoritme a jeho úspešných aplikáciách. Pred viac ako 12 rokmi som sa po prvý raz stretol s homeopatiou. Sprvu som reagoval odmietavo, pretože jej opodstatnenia boli naivné. Až neskôr som urobil to, čo sa očakáva od každého solídneho vedca. Namiesto nezmyselných špekulácií pre a proti som začal skúšať, experimentovať. A to, čo som videl a zažil v nasledujúcich rokoch, ma presvedčilo. Mal som možnosť zhovárať sa aj s viacerými úspešnými známymi osobnosťami[5] tejto alternatívnej medicíny. Ich skúsenosti so státisícami pacientov potvrdili, že najčastejšou reakciou na použitie homeopatických prostriedkov je prechodné zhoršenie stavu a že práve toto prechodné zhoršenie stavu je potvrdením správnej voľby homeopatického prostriedku. Práve táto skutočnosť veľmi pripomína proces optimalizácie simulovaným žíhaním.

Je ale na čase zasvätiť čitateľa, ktorý s homeopatiou nemá nijaké skúsenosti, do tejto alternatívnej liečebnej metódy. Objavil ju pred vyše 200 rokmi nemecký lekár Hahnemann, ktorého trápilo množstvo rôznych negatívnych vedľajších účinkov liekov. Položil si otázku, či sú takto spôsobené škody na ľudskom organizme naozaj nevyhnutným sprievodným javom liečebného procesu. Jeho prvý pokus o zmiernenie vedľajších účinkov liečebných preparátov viedol k ich riedeniu. Žiaľ, so stupňom riedenia klesali nielen vedľajšie účinky, ale i účinnosť liekov. Pri týchto pokusoch objavil jednu zvláštnu metódu riedenia a nikto netuší, ako na ňu prišiel. Zriedil liek v pomere $1 : 100$ s vodou, potom zmes desaťkrát intenzívne pretrepal. Po 30 opakovaniach tejto procedúry dostaneme takzvanú potenciu C30 pôvodného prostriedku. Z kombinatoriky už vieme vypočítať, že vzniknutý homeopatický prostriedok potencie C30 obsahuje len zlomok veľkosti $\frac{1}{100^{30}}$ pôvodného lieku. Keď začneme riedenie s 10^{21} molekulami lieku, neobsahuje C30 potencia prakticky žiadnu molekulu pôvodnej látky.

[5] Napríklad s držiteľom alternatívnej Nobelovej ceny pánom Vithoulkasom.

Táto kombinatorická úvaha je najčastejší ale aj najnaivnejší argument odporcov homeopatie, ktorí ju obviňujú z nevedeckosti. V tejto knihe sme sa ale naučili, že slovo „nemožné“ by sme nemali len tak ľahko vysloviť. Samozrejme, že homeopatické prostriedky nemôžu pracovať na tej istej chemickej báze ako bežné lieky. Ale to je to, o čo sa Hahnemann vlastne usiloval. Zostáva teda nasledujúca otázka: Ako účinkujú homeopatické lieky? Na túto otázku nepoznáme uspokojivú odpoveď. Ja pokladám za možné, že ide o čistý prenos informácie. Existujú výskumy v štruktúrach, ktoré môžu vytvárať molekuly vody. Rôznorodosť kombinácií je taká veľká, že v pohári vody možno zapamätať viac informácií ako v pamätiach všetkých počítačov na zemeguli. Existuje hypotéza o schopnosti vody zapamätať si prostredníctvom štruktúr medzi jej molekulami informácie o látkach s ktorými sa dostala do kontaktu. Pretriasanie by malo proces zapamätávania obzvlášť zintenzívniť a viesť k tomu, že vytvorené štruktúry vykazujú vysokú stabilitu.

V súčasnom štádiu vývoja vedy nie sme schopní poskytnúť na fyzikálnochemickej úrovni dôkazy v prospech alebo neprospech homeopatie. Toto zvládneme aj pre klasické lieky len zriedkavo a čiastočne. Zostáva nám len možnosť experimentovať. Typickými experimentmi v medicíne je porovnávanie účinkov lieku s takzvaným placebom. Pacienti sú náhodným spôsobom rozdelení do dvoch rovnako veľkých skupín. Jedna skupina dostane skúmaný liek, druhá čistý cukor, nazývaný placebo. Pacienti o tom ale nie sú informovaní. Po uskutočnení pokusu porovnáme účinok lieku so stupňom účinku placeba. Napríklad pri skúmaní jedného konkrétneho antidepresíva dosiahol tento liek účinnosť 43% (pomohlo 43 pacientom zo 100), pričom placebo pomohlo 30% pacientov. Prekvapila vás vysoká účinnosť placeba? Zrejme už len informácia, že sa niekto zaoberá liečením pacienta, môže spôsobiť zázraky. Poznáme množstvo lekárskych správ, ktoré dokumentujú spontánne vyliečenie ťažkých chorôb ako rakovina len vďaka placebo efektu. Žiaľ, nepoznám nijaké uspokojivé rozsiahle porovnávacie štúdie medzi homeopatickými prostriedkami a placebom. Poznáme aspoň dva dôvody, prečo je to tak. Homeopatia je síce široko praktizovaná, ale chýbajú vedecké inštitúcie, ktoré by sa bez predsudkov tejto problematike profesionálne venovali. Po druhé porovnávacie experimenty s placebom nie sú v prípade homeopatie až také jednoduché, pretože samotná choroba neurčuje vhodný homeopatický prostriedok. Na určenie vhodnej homeopatie musíme najprv určiť aj typ pacienta a v hre sú stovky rôznych typov tejto klasifikácie. Voľba vhodného homeopatického prostriedku je založená na skúsenostiach a intuícii a homeopatický lekár často potrebuje niekoľko pokusov na nájdenie toho vhodného. Vzhľadom

na ťažkosti, ktoré sme uviedli, sa už vôbec nebudeme zaoberať otázkami do akej miery sme schopní merať pri rôznych chorobách zlepšenie zdravotného stavu.

Navyše podľa niektorých predstáv optimalizuje homeopatický prostriedok len isté parametre zdravotného stavu. Pri ťažkých chronických chorobách je preto nevyhnutné podávať v určitom poradí viacero homeopatických prostriedkov počas dlhšieho obdobia.

Čo ostáva povedať na konci kapitoly, aby sa to hodilo aj na záver knihy? Vedci v základnom výskume zažili vždy veľa dobrodružstiev a tí dnešní nie sú a nebudú výnimkami. Pionierske časy nie sú len za nami, ale aj pred nami. Či v kryptológii, v počítaní s využitím náhody, DNA, kvantových výpočtoch alebo liečení pomocou „predpísania“ informačných správ organizmu. Určite ešte môžeme zažiť zázraky, ktoré uvedú vedcov do úžasu a ako sprievodný vedľajší účinok prispejú nemerateľným spôsobom k zvýšeniu kvality nášho života.

Nóvum je najprv vysmievané, potom popierané,
až nakoniec sa berie ako samozrejmosť.

Emil Filla

1. Doslov

Informatika a všeobecné vzdelanie

Dnes sú počítače rozšírené v domácnostiach podobne ako telefón či televízor. Výrazy ako PC, internet, e-mail, alebo www patria medzi najhojnejšie používané slovíčka v súčasnej informačnej spoločnosti. Ale schopnosť úspešne používať počítač alebo surfovať po internete ešte z nikoho nespravilo informatika, rovnako ako riadenia motorového vozidla z nikoho neurobilo strojára, alebo používanie prístrojov v domácnosti z nikoho neurobilo fyzika. Napriek tomu sa vyučovanie informatiky na školách príliš zaoberá sprostredkovaním vedomostí s krátkou životnosťou, napríklad o rôznych softvérových produktoch a ich používaní. Ako by vyzeralo vyučovanie fyziky, keby sme sa namiesto učenia fyzikálnych zákonov zaoberali ovládaním prístrojov a zariadení, ktorých funkčnosť je založená na princípoch fyziky? V mnohých krajinách sa vďaka tendenciám učiť „lacnú“ informatiku považuje na školách informatika za povrchný a nudný predmet, vhodný nanajvýš pre pár hackerov alebo ako náhrada písacieho stroja pri vytváraní dokumentov. V niektorých krajinách ju vďaka tomu vylúčili z osnov stredoškolskej výučby aj preto, že jej náročnosť v porovnaní s inými predmetmi bola taká nízka, že pre školy bola maturitná skúška z informatiky pod úrovňou, pod ktorú by učitelia boli ochotní ísť. Preto bolo cieľom tejto knižky, a je aj cieľom tohto doslovu, upozorniť na skutočnosť, že informatika ako vedná disciplína má netriviálnu hĺbku a prináša poznanie, ktoré mení náš pohľad na svet a stáva sa súčasťou nášho kultúrneho a duchovného bohatstva. Ak chceme učiť informatiku ako predmet na stredných školách, musíme najprv pochopiť jej prínos na úrovni hlbokých myšlienok, pojmotvorby, odhalených zákonitostí spracovania informácií a nie na úrovni každodenných aplikácií akým je telefonovanie mobilnými telefónmi. Len a len z tejto úrovne môžeme odvinúť stredoškolské plány výučby, ktoré by z informatiky spravili predmet s vážnosťou porovnateľnou s matematikou a prírodnými vedami.

Tak ako napríklad fyzika alebo biológia odvíja svoje poslanie v školách od podstaty svojej disciplíny a ich cieľov, potrebujeme sa najprv zamyslieť nad otázkou: **Čo je informatika?** Najčastejšie nachádzame nasledujúcu definíciu:

> *„Informatika je veda o algoritmickom spracovaní, reprezentácii, zapamätávaní a prenose informácií.“*

Aj keď táto zvyčajne akceptovaná definícia informatiky predstavuje algoritmy a informáciu ako hlavné objekty výskumu v informatike, nehovorí veľa o podstate informatiky a jej metódach. Pre pochopenie podstaty informatiky je oveľa dôležitejšia nasledujúca otázka:

> *„Ku ktorým vedám možno priradiť informatiku? Je metavedou ako filozofia alebo matematika, humanitná veda, prírodná veda alebo technická veda?“*

Odpoveď na túto otázku objasňuje nielen objekt skúmania, ale pomáha opísať aj jej metódy a prínos. Odpoveďou je, že informatika sa nedá jednoznačne priradiť do nijakej z uvedených kategórií vedeckých disciplín. Informatika v sebe integruje aspekty metavedy, prírodnej vedy a aj technickej vedy. Skúsime to podrobnejšie zdôvodniť v nasledujúcom texte.

Tak ako filozofia a matematika, študuje informatika všeobecné kategórie ako sú:

> *determinovanosť, nedeterminovanosť, náhoda, výpočet, informácia, pravda, jazyk, zložitosť, dôkaz, vedomosti, komunikácia, aproximácia, algoritmus, simulácia, atď.*

a prispieva k hlbšiemu chápaniu ich významu. Viacerým z týchto kategórií dala informatika nový obsah a význam.

Ako prírodná veda študuje informatika, na rozdiel od filozofie a matematiky, konkrétne existujúce objekty a procesy, určuje hranicu medzi možným a nemožným a skúma kvantitatívne zákony procesov spracovania informácií. Prírodné vedy modelujú realitu, svoje modely analyzujú a overujú ich spoľahlivosť pomocou experimentov. Všetky tieto aspekty prírodných vied nachádzame aj v informatike. Objekty sú informácie vo forme dát a algoritmy (programy, počítače) a študované procesy sú fyzikálne existujúce procesy spracovania informácií. Skúsme to dokumentovať na vývoji informatiky. Historicky prvá otázka informatiky bola nasledujúca otázka filozofického významu:

> *Existujú dobre definované úlohy, ktoré nemožno automaticky*

(pomocou počítača, nezávisle od výkonosti dnešných, či budúcich počítačov) riešiť?

Úsilie zodpovedať túto otázku viedlo k oddeleniu informatiky ako samostatnej vednej disciplíny. Odpoveď na túto otázku je kladná a dnes poznáme množstvo praktických úloh, ktoré by sme radi vedeli riešiť automaticky pomocou algoritmov, ktoré nie sú ale algoritmicky riešiteľné[6]. Už vieme, že to nie je len preto, že nik doteraz nenašiel algoritmus na ich riešenie. Podstatné je, že vieme matematicky dokázať neexistenciu takýchto algoritmov.

Po klasifikácii úloh na algoritmicky riešiteľné a algoritmicky neriešiteľné sa ústrednou otázkou informatiky stáva nasledujúca prírodovedná otázka: **„Aké ťažké sú konkrétne algoritmické úlohy?“**

Obťažnosť úlohy ale nemeriame intelektuálnou obťažnosťou nájdenia algoritmu na jej riešenie. Obťažnosť úlohy sa meria množstvom nevyhnutnej a postačujúcej počítačovej práce potrebnej na jej algoritmické riešenie. Informatici objavili, že existujú ľubovoľne ťažké úlohy, dokonca také ťažké, že na ich riešenie by nestačila ani celá energia vesmíru. Takže vieme, že existencia algoritmu pre danú úlohu ešte nezaručuje jej automatizovateľnosť v praxi.

Úsilie rozdeliť úlohy na **prakticky riešiteľné** a **prakticky neriešiteľné** viedlo a ešte stále vedie k najfascinujúcejším objavom informatiky.

Z knižky môžeme spomenúť efekty, keď malá zmena v požiadavkách na riešenie úlohy spôsobila skok z fyzikálne nerealizovateľného množstva práce na niekoľkosekundovú záležitosť na bežnom PC. Najlepšie sme to videli na randomizovaných algoritmoch, kde sme sa vzdali hypoteticky 100% záruky korektnosti deterministických algoritmov a navrhli sme randomizované algoritmy, ktoré exponenciálne redukovali zložitosť. Pritom pravdepodobnosť chyby týchto algoritmov je menšia ako 1 k veku vesmíru v sekundách, a teda z praktického hľadiska nielen akceptovateľná, ale aj menšia ako pravdepodobnosť hardvérovej chyby v priebehu výpočtu 100% spoľahlivého deterministického algoritmu.

Vďaka takýmto kvantitatívnym skokom dokáže informatika zázraky, pomocou ktorých môžeme nečakane riešiť zdanlivo bezvýchodiskové situácie. Vplyv týchto divov na prax je neuveriteľný. Napríklad výsledky o kvalitatívnych zákonitostiach výpočtov (výpočtovej zložitosti) viedli k novej definícii bezpečnosti v kryptografii a cez ňu k moderným kryptosystémom s verejným kľúčom (public-key), bez ktorých si nevieme dnes vôbec

[6] Pozri 4. kapitolu.

predstaviť e-komerciu alebo online banking. Je vôbec ospravedlniteľné vyučovať kompetencie narábania so špecifickými softvérovými systémami namiesto kreatívnych, hlbokých myšlienok, ktoré boli naozaj tým, čo zmenilo dnešný svet?

Napriek uvedeným prírodovedným aspektom informatiky je informatika pre väčšinu absolventov štúdia inžinierskou disciplínou, ktorá obsahuje aspekty technických vied, ako sú:

> *organizácia procesu vývoja, modelovanie, opis, formulovanie cieľov, testovanie, kontrola kvality, kompatabilita s existujúcimi systémami, atď.*

K tomu patria aj manažérske aspekty ako sú

> *organizácia tímov a riadenia projektu, odhad ceny, plánovanie, produktivita, odhad časových rámcov a termínov, marketing, uzatváranie zmlúv, atď.*

Ako komplexné sú tieto úlohy si môžeme uvedomiť, keď uvážime, že mnohé softvérové produkty sú zostavené z miliónov počítačových príkazov a boli vyvíjané a vylepšované množstvom špecialistov počas mnohých rokov. Ak tieto programy neboli systematicky vyvíjané a dokumentované, nie je prakticky možné overiť ich správne fungovanie, hľadať v nich chyby, alebo ich dopĺňať o nové funkcie. Pri takejto praktickej zložitosti reálnych systémov nemožno s matematickou presnosťou všetko modelovať a predpovedať. Dôsledkom čoho sa veľa pragmatických rozhodnutí uskutoční len na základe intuície a skúseností tvorcov. Nezdá sa vám teraz štúdium informatiky dostatočne ťažké? Zvládnuť netriviálnu matematiku pri riešení úloh a umenie budovať pragmaticky komplexné systémy spája dva veľmi odlišné spôsoby myslenia. Na jednej strane je to analytický matematicko-prírodovedný spôsob myslenia a na druhej strane presne nesformalizované postupy technických vied s množstvom improvizácie založenej na skúsenostiach. Ale práve spojenie týchto dvoch jazykov a spôsobov myslenia v jednom študijnom odbore je tým, čo dáva tomuto štúdiu najväčšiu perspektívu.

Vráťme sa k otázke, čo a ako vyučovať v predmete informatika na stredných školách. Vzdajme sa prístupu od „farebných aplikácií k nejakým vedomostiam“ o tom, ako príslušné softvérové produkty pracujú a ako ich používať. Objav parnej lokomotívy bol tiež fascinujúci a vyučovanie fyziky nezačíname tým, že sa učíme ovládať riadiť lokomotívu. Dnes sú najrozličnejšie aplikácie na obrazovke tak bežné, že nevyvolávajú nijaké nadšenie. Zato naša dlhoročná prax vyučovania na stredných školách stále

potvrdzuje, že mladých ľudí možno nadchnúť práve sprostredkovaním vedomostí, znovuobjavovaním a objasňovaním zákonitostí a učením sa ich inteligentne využívať na riešenie neľahkých úloh.

Ako začať? Tak ako matematika nemôže začať s oslňujúcimi ideami ale naopak s jednoduchým počítaním, musí si aj informatika zvoliť zodpovedajúci začiatok. Základnou schopnosťou informatika v tomto zmysle je vedieť programovať. Nie v zmysle syntaktického ovládania nejakého programovacieho jazyka, ale v zmysle schopnosti riešiť malé úlohy pomocou programov na počítači. Učiť sa programovať v sebe zahŕňa oveľa viac ako si mnohí uvedomujú. Programovaním sa učíme komunikovať s technikou. A to s technikou bez inteligencie a tým aj bez schopnosti improvizovať. To nás núti komunikovať tak, aby sme boli bezpochýb správne pochopení. Musíme byť schopní opísať činnosti tak jednoznačne, že neostane priestor na neželanú chybnú interpretáciu. Už táto skutočnosť obohacuje schopnosť vyjadrovať sa v zmysle extrémnej presnosti a rozvíja zodpovedajúci spôsob myslenia. Ďalším, nie pre každého očividným, prínosom je budovanie mostov medzi matematickým myslením a inžinierskym prístupom. Opis a analýza úlohy, hľadanie algoritmu na jej riešenie je silne spojené so schopnosťou matematického analytického myslenia a tiež s používaním presného jazyka matematiky na opis úlohy a cesty na jej riešenie. Samotné programovanie a testovanie navrhnutého algoritmu je do značnej miery inžinierska práca. Hľadanie chýb a možných vylepšení programu vzhľadom na rôzne parametre zaberie zvyčajne niekoľkonásobne viac času, ako vyhotovenie prvého prototypu. Programovanie prináša radosť zo samostatného absolvovania cesty od zadania, až k požadovanému cieľu. Učiteľa stavia skôr do úlohy pomocníka ako kritika, pretože žiaci si môžu overovať správnosť svojich programov samostatne, testovaním na skutočných údajoch a programy korigovať až do vytvorenia požadovaného produktu. Vyučovaním programovania sprostredkujeme aj modulárnu metódu návrhu zložitých technických systémov. Pri komplexnejších úlohách vytvoríme najprv programy pre malé čiastočné úlohy a overíme ich korektnosť. Tieto programy použijeme ako moduly, z ktorých skladáme postupne zložitejšie a zložitejšie moduly.

Už pri výučbe programovania začíname s pojmotvorbou. Učíme sa, čo je program, údaje. Učíme sa, ako pracuje počítač, ako sú reprezentované a uložené údaje v počítači. Na tomto základe ďalej môžeme budovať špecifikovaním ďalších kľúčových pojmov a špecifikovaním základných konceptov. Ako sa to dá realizovať, sme načrtli v tejto knihe. Pritom nie je až také dôležité, ktoré oblasti informatiky, ako sú databázy, kryptografia, vizualizácia, návrh obvodov alebo automatov, algoritmika, komunikačné

siete, numerické algoritmy vo vedeckých výpočtoch alebo iné si zvolíme. Dôležité je, aby sme boli schopní sprostredkovať základné koncepty ako algoritmus, výpočtová zložitosť, logika a korektná argumentácia, verifikácia a testovanie, modulárny návrh systémov, simulácie, rekurzia, atď.

Zavedenie informatiky do vyučovania na stredných školách nie je len problematická úloha, ale v prvom rade šanca. Je to šanca priniesť do všeobecného vzdelania paradigmy sveta procesov spracovania informácií. Je to šanca zatraktívniť matematiku, jej inštrumentalizovanie pri riešení praktických úloh. Je to šanca postaviť mosty k technickým vedám a priblížiť inžiniersky spôsob práce žiakom na stredných školách. A je to šanca vykročiť smerom k dlho proklamovanej interdisciplinárnosti, ktorá sa v informatike realizuje prirodzeným spôsobom.

Veda, ktorá by bola vybudovaná len za praktickým účelom, nie je možná. Pravdy sú užitočné len vtedy, keď sú navzájom poprepájané. Ak hľadáš len tie pravdy, z ktorých sa dajú očakávať priame dôsledky pre prax, strácajú sa väzby medzi ohnivkami reťazce a reťaz sa roztrhne.

Jules Henri Poincaré

2. Doslov

Je základný výskum luxusom alebo existenčnou nevyhnutnosťou?

Môže si, resp. má si nejaká krajina dovoliť investovať do základného výskumu? Sú držitelia Nobelových a iných vedeckých cien zbytočným luxusom? Nestačí nám aplikovaný výskum na zabezpečenie dlhodobého blahobytu a pokroku? Čo sa stane, ak sa vzdáme základného výskumu? Má byť mierou investovania do výskumu odhad prínosu predaja vyvinutých produktov na medzinárodnom trhu?

Počas svojho pôsobenia na rôznych univerzitách som sa stretával s týmito otázkami veľmi často. Žiaľ práve z politických a manažerských kruhov pochádzali krátkozraké odpovede a hlúpe rozhodnutia. Volanie po takzvanom ekonomickom plánovaní vedy, ktorá spočíva vo vyčíslení a vyžadovaní ekonomického prínosu vložených investícií do vedy v priebehu pár rokov, patrí do kategórií veľmi obmedzeného chápania úlohy vedy v spoločnosti a jej prínosu pre jej blahobyt. Populistické otázky, či základný výskum vôbec prispieva k rastu alebo udržaniu životnej úrovne a či to vôbec je zmysluplné využitie peňazí daňových poplatníkov a volanie po meraní užitočnosti vedy ekonomickým ziskom, nedokážem lepšie komentovať ako slovami Louisa Pasteura: „Nešťastníci sú tí, ktorým je všetko jasné“.

Najprv si musíme uvedomiť, že neexistuje jasná hranica medzi aplikovaným a základným výskumom. Aby sme ale vôbec túto problematiku dokázali pochopiť, musíme sa najprv zaoberať úlohou vedy v dnešnej tzv. znalostnej spoločnosti a v dejinách ľudstva. Položme si najprv otázku: Čo je aplikovaný výskum a čo je teoretický základný výskum a nachádzajú sa tieto oblasti vôbec vo vzájomnej konkurencii? Súčasný vývoj

ukazuje, že máme čoraz väčší počet výskumných projektov, ktoré sa nedajú jednoznačne zaradiť len do jednej z týchto kategórií. Výskum génov či konštrukcia kvantového počítača patria do základného výskumu, a jednako majú prakticky okamžité komerčné uplatnenia. Existuje veľa príkladov z kryptológie, spoľahlivej komunikácie, automatického rozpoznávania reči, logistiky a optimalizácie riadenia pracovných procesov, návrhu sietí atď., v ktorých objavy teoretického výskumu zakrátko celkom zmenili prax. Tu niet jasnej hranice medzi základným a aplikovaným výskumom.

No zdá sa, že autori návrhov šetrenia na luxuse základného výskumu vedia, v čom väzí rozdiel medzi základným a aplikovaným výskumom. Aplikovaný výskum je ten, v ktorom môžeme už počas plánovania vyčísliť ekonomický prínos. Ak sa to dá, musíme si položiť otázku, či v takomto prípade vôbec ide o výskum. Posudzoval som množstvo takto naplánovaných projektov a všetky mali jedno spoločné. V skutočnosti ich nebolo treba. Firmy mohli hravo dosiahnuť vytýčené ciele zamestnaním absolventov univerzít s patričným know-how. A to je jadro nášho problému. To, čo sa v procese riadenia často deklaruje ako aplikovaný výskum, je iba rutinné uplatnenie znalostí, pri ktorom nevzniká nič pôvodné, nijaké nové poznatky.

Rast vedomostí je najdôležitejším predpokladom dlhodobého pokroku. Kto predvídal pred 25 rokmi éru internetovej komunikácie a internetu, e-komercie a online bankingu, ktoré tak výrazne zmenili celú spoločnosť, zefektívnili množstvo jej procesov a prispeli k ťažko vyčísliteľnému rastu životnej úrovne? Kto pred 25 rokmi tušil, že komunikačná technika sa stane dôležitým odvetvím priemyslu? Mohol vtedy vôbec niekto odhadovať finančnú a ekonomickú prosperitu vtedajšieho výskumu? Veda, ktorá toto umožnila, spočívala v prvom rade na základnom výskume fyzikálnej povahy pri objavovaní nových materiálov pre komunikačné technológie a v matematike a informatike pri vývine efektívnych a spoľahlivých algoritmov. Len veľmi málo ľudí má predstavu, aké hlboké poznatky a objavy týchto vedných disciplín boli nevyhnutné na vývin súčasného komunikačného luxusu. Nijaké zmeny vedúce k zvýšeniu blahobytu spoločnosti nie sú možné bez predchádzajúcej akumulácie poznatkov. Iba konzumovať súčasné poznatky môže prinášať zisky len veľmi krátkodobo. Bez nepredpovedateľného generovania nových poznatkov niet životného napredovania ani blahobytu.

Slovíčko blahobyt nie je v tejto súvislosti zvolené práve šťastne. Niektorí politici a ekonómovia sa nám usilujú nahovoriť, že blahobyt je najdôležitejší cieľ spoločnosti. Tento populistický názor je prirodzene obľúbený,

pretože sa týka každého a každý sa chce mať lepšie. Ale argumentovať len z pozície súčasného blahobytu je rovnako nebezpečné, ako keď niekto v dobe kamennej odporúčal, každý deň sa čo najlepšie najesť a neodkladať zásoby na zimu. Tento spôsob uvažovania nepamätá na to, že v zime možno zomrieť od hladu. Ak sa nazdávame, že tieto problémy patria minulosti, odpovedám vám, že sa na svet pozeráte cez ružové okuliare.

Z môjho pohľadu bežíme stále s časom o preteky a rozhoduje sa o existencii ľudstva. Naša zemeguľa nám dožičila 10 000 rokov mimoriadne priaznivých klimatických podmienok, ktorým v značnej miere vďačíme za dynamický vývin našej civilizácie. Nemáme ale ani potuchy, čo všetko nás ešte čaká. Keď prídu časy plné problémov neznámych rozmerov, nebude rozhodujúce, či žijeme v blahobyte alebo nie, ale či sme dospeli do štádia vývoja, v ktorom by sme boli schopní tieto problémy zvládnuť. To môže byť naozaj existenčná otázka. Spomeňte si na obrovskú vlnu tsunami pred pár rokmi, ktorá stála život desaťtisíce ľudí. Stav našich poznatkov nebude rozhodovať len o tom, či sme schopní s dostatočným predstihom ďalšiu takúto tsunami predpovedať, ale či sme schopní v dostatočnej miere zabrániť škodám, ktoré by mohla spôsobiť. Preto je neospravedlniteľne riskantné sústrediť sa v časoch hojnosti a nadbytku na udržovanie a budovanie blahobytu namiesto investícii do ďalšieho vývinu.

Transformácia vedomosti do prakticky užitočných technológií je jedna z dôležitých mier pokroku. Potrebné vedomosti vznikajú príznačne vo výskume, ktorého ekonomickú užitočnosť v nasledujúcich 10 až 20 rokoch nevieme odhadnúť. Deň držiteľov Nobelových cien počas osláv 150. výročia ETH Zürich poukázal na skutočnosť, že cesta od základných objavov vedy k priemyselným aplikáciám trvá v priemere 20 rokov. A kto sa odváži dnes predpovedať, ako bude o 20 rokov vyzerať náš svet? Azda len tí šťastní nevedomí, ktorí sa nazdávajú, že rozumejú všetkému.

Požadovanie „čisto“ aplikovaného a „užitočného“ výskumu bez snahy o generovanie nových poznatkov je rovnako nebezpečné ako považovanie umenia za nadmerný luxus. Vraj načo sú umelci ako Picasso, Dalí, Hundertwasser, Mozart alebo Beethoven? Nestačia väčšine k spokojnosti reprodukcie a populárna hudba? Nie sú veľkí spisovatelia niečím, bez čoho môže ľudstvo pokojne existovať? Veď koľko sa predá aj málohodnotných románov. Pred skutočnosťou, že veľké umelecké diela ovplyvnili vývoj celých generácií, zavrime radšej oči. A rovnako zabudnime na to, že vede a umeniu bez ohraničeného horizontu vďačíme nielen za vysnený blahobyt, ale aj za celé napredovanie našej civilizácie.

Cieľom tohto doslovu nebolo navrhnúť proporcie medzi investíciami do ap-

likovaného a základného výskumu. Chceli sme len poukázať na to, aké nebezpečné je redukovať financovanie vedy na podporu projektov s vyčísliteľným úžitkom. Takéto správanie sa je extrémne krátkozraké a vedie k strate postavenia produkcie znalostí a celého vzdelávania v spoločnosti. Spoľahnúť sa na samoriadenie vedeckého výskumu prostredníctvom priemyselného dopytu prináša so sebou riziko stagnácie vývoja. Ak nejaký podnik investoval veľa peňazí do nejakej technológie, nemá záujem o podstatné technologické zmeny, pokiaľ sa mu ešte pôvodné investície nezhodnotili. Len konkurencia tlačí firmy k podstatným inováciám. Jediný mne známy funkčný prístup prerozdeľovania prostriedkov na vedu je prostredníctvom posudkov nezávislých osobností vedy. Hľadať akékoľvek kvantitatívne meradlá na posudzovanie kvality kreatívnej výskumnej práce je nezmyselné. Je škoda, že Slovensko ešte nevedelo urobiť podstatný krok od celoplošného socialistického rozdeľovania financií v školstve a vede na prerozdeľovania prostriedkov na základe takýchto posudkov zahraničných expertov. Priveľa slovenských inštitúcií má strach z nastavenia zrkadla. Prosperitu vo vede môže priniesť len prerozdelenie prostriedkov tak, aby kreatívne vedecké pracoviská mali zabezpečené dostatočné prostriedky na sledovanie náročných cieľov. Súčasná rovnostárska polonasýtenosť finančnými prostriedkami na vedeckých inštitúciách bez ohľadu na ich skutočné kvality je tragédiou riadenia slovenskej vedy v celom období od vzniku Slovenskej republiky.

Aké je naše posolstvo neohraničené slovenskými pomermi? Čím viac dobrej vedeckej práce, tým istejšia budúcnosť, tým viac kreatívnej práce a tým viac vnútornej spokojnosti. Ja osobne verím, že kreativita je zmyslom nášho života, a preto ukončím doslov nasledujúcimi citátmi:

> Tvoriac čakal som na uzdravenie, tvoriac som vyzdravel.
>
> Heinrich Heine

> Korene všetkého zla sú v nevedomosti.
>
> Buddha

> Zmysel života vidím v tvorbe, ktorá je nekonečná.
>
> Maxim Gorkij

Pôvodná verzia tohto doslovu vyšla v nemčine ako článok „io new management“, Nr. 10/2006.

Literatúra

[BB03] D. Bongartz and H.-J. Bäckenhauer. *Algorithmische Konzepte der Bioinformatik.* Teubner, 2003.

[Beu02a] A. Beutelspacher. *Geheimsprachen. Geschichte und Techniken.* Verlag P. H. Beck, München, 2002.

[Beu02b] Albrecht Beutelspacher. *Kryptologie.* Friedr. Vieweg & Sohn, Braunschweig, sixth edition, 2002. Eine Einführung in die Wissenschaft vom Verschlüsseln, Verbergen und Verheimlichen. [Úvod do vedy šifrovania, ukrývania a utajovania].

[Čer85] V. Černý. A Thermodynamical Approach to the Traveling Salesman Problem: An Efficient Simulation Algorithm. *Journal of Optimization Theory and Applications*, 45, 1, 41–51, 1985.

[DH76] Whitfield Diffie and Martin E. Hellman. New directions in cryptography. *IEEE Trans. Information Theory*, IT-22(6):644–654, 1976.

[Die04] Martin Dietzfelbinger. *Primality testing in polynomial time*, volume 3000 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, 2004. From randomized algorithms to "PRIMES is in P".

[DK02] Hans Delfs and Helmut Knebl. *Introduction to cryptography.* Information Security and Cryptography, Principles and applications. Springer-Verlag, Berlin, 2002.

[Drl92] K. Drlica. *Understanding DNA and Gene Cloning. A Guide for CURIOUS.* John Wiley and Sons, New York, 1992.

[Fey61] R. P. Feyman. Miniaturisation. *D.H. Gilber (ed)*, pages 282–296, 1961.

[Hro97] J. Hromkovič. *Communication complexity and parallel computing.* Texts in Theoretical Computer Science. An EATCS Series. Springer-Verlag, Berlin, 1997.

[Hro04a] J. Hromkovič. *Algorithmics for Hard Problems.* Springer Verlag, 2004.

[Hro04b] J. Hromkovič. *Randomisierte Algorithmen. Methoden zum Entwurf von zufallsgesteuerten Systemen für Einsteiger.* Teubner, 2004.

[Hro04c] J. Hromkovič. *Theoretische Informatik.* Teubner, 2004.

[Hro05] J. Hromkovič. *Design and analysis of randomized algorithms, Introduction to design paradigms.* Texts in Theoretical Computer Science. An EATCS Series. Springer-Verlag, Berlin, 2005.

[KGV83] S. Kirkpatrick, C. D. Gelatt Jr., M. P. Vecchi. Optimization by Simulated Annealing, *Science*, 220, 4598, 671-680, 1983.

[KN97] Eyal Kushilevitz and Noam Nisan. *Communication complexity.* Cambridge University Press, Cambridge, 1997.

[MR95] Rajeev Motwani and Prabhakar Raghavan. *Randomized algorithms.* Cambridge University Press, Cambridge, 1995.

[MRR+53] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, Teller A. H., and E. Teller. Equation of state calculation by fast computing machines. *Journal of Chemical Physics*, 21:1087–1091, 1953.

[PRS05] Ch. Păun, G. Rozenberg, and A. Salomaa. *DNA Computing. New Computing Paradigms.* Springer Verlag, 2005.

[RSA78] R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Comm. ACM*, 21(2):120–126, 1978.

[Sal96] Arto Salomaa. *Public-key cryptography.* Texts in Theoretical Computer Science. An EATCS Series. Springer-Verlag, Berlin, second edition, 1996.