ÚVOD DO TEÓRIE ZLOŽITOSTI

RNDr. Ivana Černá, CSc.
Fakulta informatiky MU, Brno

20. listopadu 1997

Tento učebný text je úvodom do teórie zložitosti.

Počiatky teórie zložitosti sa datujú do 60–tych rokov, keď s rozširujúcim sa využívaním počítačov vyvstala otázka efektivity jednotlivých programov. Jej cieľom je dať odpoveď na otázku, prečo riešenie niektorých problémov na počítači je tak náročné. Podobne ako teória vyčísliteľnosti vedie k rozdeleniu problémov na spočítateľné (vyčísliteľné, algoritmicky riešiteľné) a nespočítateľné, tak teória zložitosti sa snaží klasifikovať vyčísliteľné problémy ako prakticky riešiteľné (feasible computable) a prakticky neriešiteľné (intractable). Teória výpočtovej zložitosti zaznamenala v uplynulých rokoch pozoruhodný vývoj. Vďaka nej sa podarilo nájsť efektívnejšie metódy riešenia mnohých dôležitých problémov alebo naopak preukázať, že daný problém je natoľko zložitý, že ho v praxi nie je možné v plnom rozsahu zvládnuť.

V teórii zložitosti môžeme vymedziť tri najdôležitejšie oblasti rôzneho charakteru. Prvá sa týka presného definovania pojmov algoritmus a zložitosť algoritmu. Definuje sa pojem výpočtového modelu, ako formalizácia pojmu algoritmus, a pojem zložitosti výpočtu. Zložitosť výpočtu je veľkosť výpočtových zdrojov (najčastejšie času a pamäti) potrebných na vykonanie výpočtu na uvažovanom modele. V skutočnosti bolo navrhnutých niekoľko veľmi rozdielnych výpočtových modelov—od čiste teoretických, akými sú Turinogve stroje, až po viac–menej realistické, ako napr. rôzne varianty RAM-ov. Otázky nakoľlo ovplyvní výber konkrétneho model získané výsledky a nakoľko tieto odrážajú skutočnosť, tiež patria do opisovanej oblasti teórie zložitosti.

Ak nejakým spôsobom ohraničíme výpočtové zdroje (čas, pamäť) uvažovaného výpočtového modelu, zúži sa množina riešiteľných problémov a dostávame sa k pojmu zložitostná trieda. Zložitostné triedy poskytujú elegantný nástroj umožňujúci klasifikáciu problémov podľa ich výpočtovej obtiažnosti a ich rozdelenie na prakticky zvládnuteľné a nezvládnuteľné problémy. Príslušnosť problému do konkrétnej zložitostnej triedy charakterizuje jeho vnútornú obtiažnosť. Táto oblasť zahŕňa dôležitý pojem „negatívneho výsledku“: t.j. tvrdenia, že isté problémy nie sú s určitými výpočtovými zdrojmi riešiteľné. Inak je možné tento problém sformulovať aj ako otázku, či určité zložitostné triedy sú rôzne alebo nie. Problémy spadajúce do tejto oblasti patria k vôbec najťažším problémom riešeným v teórii zložitosti.

Tretia oblasť sa sústreďuje na problémy o ktorých vieme (alebo predpokladáme), že sú prakticky nezvládateľné. Teória zložitosti skúma spôsoby, ako ich riešiť použitím iných výpočtových paradigiem (ako sú paralelizmus, náhodnosť, aproximácie, heuristiky) a aké sú hranice ich použiteľnosti.

Obsah

Kapitola 1

Základné pojmy

Cieľom tejto časti je definovať základné pojmy, ktoré sa v učebných textoch vyskytujú, a predovšetkým fixovať používané značenia.

Abeceda, slovo, jazyk. Usporiadanie slov. Kódovanie slov.

Definícia 1.1 *1.* Abecedou *budeme nazývať ľubovoľnú konečnú množinu symbolov.*
2. Konečná postupnosť symbolov nad abecedou Σ sa nazýva slovo. *Uvažujeme aj prázdne slovo, označujeme ho symbolom λ.*
3. Dĺžku *slova w označujeme $|w|$.*
4. Symbol Σ^n používame na označenie množiny všetkých slov dĺžky n nad abecedou Σ, množinu všetkých slov nad abecedou Σ značíme Σ^*. Podmnožina Σ^*, t.j. ľubovoľná množina slov (nad abecedou Σ), sa nazýva jazyk *(nad abecedou Σ).*

Inými slovami: Σ^* je nekomutatívny monoid s operáciou zreťazenia a prázdnym slovom ako jednotkou. Veľká časť funkcií, s ktorými budeme v týchto učebných textoch pracovať je zobrazením zo Σ^* do Γ^*, kde Σ a Γ sú abecedy. Špeciálnym typom týchto funkcií sú homomorfizmy:

Definícia 1.2 *Nech Σ a Γ sú abecedy.* Homomorfizmus *je zobrazenie zo Σ^* do Γ^* také, že*
1. $h(\lambda) = \lambda$ *a*
2. $h(u \cdot v) = h(u) \cdot h(v)$.

Každý homomorfizmus je jednoznačne definovaný obrazmi prvkov zo Σ. Využitím homomorfizmus môžeme vytvárať "preklady" medzi rôznymi abecedami. Pre dané dve abecedy Σ a Γ , slovo nad abecedou Σ môžeme previesť na slovo nad abecedou Γ takto: ak Σ má menej prvkov než Γ , tak definujeme injektívny homomorfizmus tak, že obrazom písmena zo Σ je písmeno z Γ; slovám nad Σ sú priradené slová nad podmnožinou Γ. V prípade, že Σ má viac prvkov než Γ, postujeme nasledovne: vyberieme prirodzené číslo n také, že $|\Sigma| \leq |\Gamma|^n$ a definujem injektívny homomorfizmus zobrazujúci každý prvok Σ na slovo dĺžky n nad Γ (predpokladáme $|\Gamma| > 1$). Slová nad abecedou Σ sme *zakódovali* slovami nad abecedou Γ. Zakódované slová môžeme jednoznačne rozkódovať tým, že ich rozdelíme na podreťazce dĺžky n.

Väčšina objektov, s ktorými pracujeme v týchto učebných textoch, sa dá popísať slovami nad nejakou abecedou: algoritmy, Turingove stroje, formuly, prípady problémov atď. Využívajúc popísané kódovanie, môžeme každý takýto objekt identifikovať so slovom nad vybranou abecedou.

Uvedený spôsob kódovania môžeme použiť aj na zakódovanie niekoľkých slov nad danou abecedou jedným slovom nad tou istou abecedou. Pretože tento spôsob kódovania budeme v ďaľšom používať veľmi často, popíšeme ho podrobnejšie. Našim cieľom je zakódovať niekoľko slov pomocou jediného tak, aby proces kódovania aj dekódovania bol jednoduchý.

Pre danú abecedu Σ fixujeme symbol $\sharp$, nepatriaci do Σ, a každej dvojici slov x, y nad abecedou Σ priradíme slovo $x\sharp y$ nad abecedou $\Sigma \cup \{\sharp\}$. Ďalej fixujeme injektívny homomorfizmus h z $\Sigma \cup \{\sharp\}$ do Σ. Kódom dvojice x, y je $\langle x, y\rangle = h(x\sharp y)$.

Napríklad, pre abecedu $\Sigma = \{0, 1\}$ môžeme zvoliť $h(0) = 00$, $h(1) = 11$ a $h(\sharp) = 01$. Je zrejmé, že navrhnuté kódovanie ako aj dekódovanie môže byť realizované konečným automatom s výstupom, a teda že je maximálne jednoduché.

Popísaný typ kódovania sa označuje aj ako *párovacia funkcia*. Vo všeobecnosti nemusí byť bijekciou a preto pri dekódovaní musíme brať do úvahy, že niektoré slová nie sú kódom žiadnej dvojice. Párovacia funkcia sa dá prirodzeným spôsobom zovšeobecniť aj na n–tice slov nasledovne: $\langle x, y, z\rangle = \langle\langle x, y\rangle, z\rangle$; $\langle x, y, z, t\rangle = \langle\langle\langle x, y\rangle, z\rangle, t\rangle$ atď.

Na množine slov môžeme definovať usporiadanie. Predpokladajme, že je dané usporiadanie prvkov abecedy Σ. V *lexikografickom usporiadaní* prvkov zo Σ^* slovo α predchádza slovo β ak buď α je prefixom (počiatočným úsekom) β, alebo prvé písmeno zľava, ktoré je rozdielne v týchto slovách, je menšie v α. (Napr. 379313 predchádza 387 a to predchádza 38782). Niekedy je výhodnejšie používať *rastúce usporiadanie*. Pri tomto usporiadaní slovo kratšej dĺžky predchádza slovo väčšej dĺžky a slová rovnakej dĺžky sú usporiadané lexikograficky.

Pre ľubovoľnú abecedu Σ, ktorej prvky sú fixne usporiadané, určuje rastúce usporiadanie bijekciu medzi množinami $\mathbb{N} \setminus \{0\}$ a Σ^*. Slovu w priradíme číslo, ktoré je jeho pozíciou (poradovým číslom) v rastúcom usporiadaní množiny Σ^*. Napríklad pre množinu $\{0, 1\}^*$ stačí, ak k danému slovu w pripojíme na začiatok symbol 1, a výsledný reťazec čítame ako prirodzené číslo zapísané v binárnej sústave. Naopak, k danému prirodzenému číslu nájdeme zodpovedajúce slovo tak, že ho zapíšeme v binárnej sústave a odstránime prvú 1.

Špeciálnym prípadom je abeceda $\Sigma = \{0, 1\}$. Slovo nad touto abecedou môžeme chápať aj ako zreťazenie symbolov, aj ako reprezentáciu celého čísla. Definujeme bijekciu medzi celými číslami a reťazcami nad abecedou $\{0, 1\}$ ako:

1. Prázdny reťazec λ reprezentuje číslo 0.
2. Reťazec $0x$ reprezentuje celé kladné číslo, ktorého binárny zápis je $1x$.
3. Reťazec $1x$ reprezentuje celé záporné číslo, ktorého absolútna hodnota má binárny zápis $1x$.

Pripomíname, že pre celé číslo y je dĺžka jeho binárnej reprezentácie približne dvojkový logaritmus jeho absolútnej hodnoty. Binárnu reprezentáciu čísla y budeme označovať $bin(y)$.

Množiny

Symbolom $\subseteq$ označujeme množinovú inklúziu. Symbol $\subset$ používame na označenie ostrej množinovej inklúzie, t.j. $A \subset B$ práve vtedy, ak A je podmnožinou B a existuje prvok množiny B nepatriaci do A. Kardinalitu množiny S (t.j. počet prvkov množiny S) označujeme $|S|$.
Množinu reálnych čísiel budeme označovať $\mathbb{R}$, množinu prirodzených čísiel (vrátane 0) $\mathbb{N}$.

Funkcie, asymptotiky

Funkcie, s ktorými sa budeme v teórii zložitosti stretávať, budú vždy celočíselné, t.j. z $\mathbb{N}$ do $\mathbb{N}$. Dokonca aj v prípadoch, keď použijeme označenie pripúšťajúce iné hodnoty než prirodzené čísla (napr. $\sqrt{n}$), budeme pod týmto označením chápať celočíslenú funkciu. Inými slovami, pod každou funkciu, ktorú označujeme $f(n)$, chápeme v skutočnosti funkciu $\lceil f(n)\rceil$, kde symbol $\lceil x\rceil$ označuje hornú celú časť čísla x. Symbol $\log n$ bude vždy označovať dvojkový logaritmus.

Nech g je reálna funkcia definovaná na množine $\mathbb{N}$. Pod označením $\mathcal{O}$ rozumieme množinu funkcií

$$\begin{aligned}\mathcal{O}(g(n)) = \{f(n) \ \mid & \text{ existujú kladné konštanty } c \text{ a } n_0 \text{ také, že} \\ & 0 \le f(n) \le cg(n) \text{ pre všetky } n \ge n_0\}\end{aligned}$$

Neformálne, funkcia $f(n)$ patrí do množiny $\mathcal{O}(g(n))$ práve ak $f(n)$ rastie tak ako $g(n)$ alebo pomalšie. Napriek tomu, že $\mathcal{O}(g(n))$ je množina, používame zápis $f(n) = \mathcal{O}(g(n))$ na vyjadrenie skutočnosti, že $f(n)$ je prvkom $\mathcal{O}(g(n))$.

Podobne ako $\mathcal{O}$ notácia poskytuje asymptotické horné ohraničenia funkcie, tak Ω notácia poskytuje asymptotické dolné ohraničenia. Formálne, pod označením $\Omega(g(n))$ rozumieme množinu funkcií

$$\begin{aligned}\Omega(g(n)) = \{f(n) \ &|\ \text{existujú kladné konštanty } c \text{ a } n_0 \text{ také, že} \\ &0 \leq cg(n) \leq f(n) \text{ pre všetky } n \geq n_0\}\end{aligned}$$

Analogicky používame označenie $f(n) = \Omega(g(n))$ na vyjadrenie skutočnosti, že funkcia $f(n)$ rastie asymptotikcy rýchlejšie než funkcia $g(n)$.

Fakt, že súčasne platí $f(n) = \mathcal{O}(g(n))$ a $f(n) = \Omega(g(n))$ označujeme ako $f(n) = \Theta(g(n))$ a hovoríme, že f a g rastú rovnako rýchlo alebo že sú asymptoticky rovnaké. Formálne, pod označením $\Theta(g(n))$ rozumieme množinu funkcií

$$\begin{aligned}\Theta(g(n)) = \{f(n) \ &|\ \text{existujú kladné konštanty } c_1, c_2 \text{ a } n_0 \text{ také, že} \\ &0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n) \text{ pre všetky } n \geq n_0\}\end{aligned}$$

Nakoniec, používame označenie $f(n) = o(g(n))$ ak funkcia f rastie asymptoticky pomalšie ako funkcia g. Formálne,

$$\begin{aligned}o(g(n)) = \{f(n) \ &|\ \text{pre každú konštatnu } c \text{ existuje konštanta} n_c \text{ taká, že} \\ &f(n) \leq cg(n) \text{ pre všetky } n \geq n_c\}\end{aligned}$$

Formuly

Ďalej pripomenieme niekoľko frekventovaných pojmov z logiky.

Booleovská premenná je ľubovoľný symbol, ktorému môže byť priradená pravdivostná hodnota 1 (označovaná aj true) alebo 0 (false).

Nech X je spočítateľná množina booleovských premenných. Trieda *booleovských formúl* (alebo len formúl) nad X je najmenšia množina definovaná nasledovne:

1. Booleovské konštanty true a false sú formuly.
2. Pre každú booleovskú premennú x z X, x je formula.
3. Ak F_1 a F_2 sú formuly, tak aj $(F_1 \wedge F_2)$, $(F_1 \vee F_2)$ a $\neg(F_1)$ sú formuly.

Ďalej definujeme triedu *kvantifikovaných booleovských formúl* nad X takto:

4. Každá booleovská formula je kvantifikovaná booleovská formula.
5. Ak x je z X a F je kvantifikovaná formula, tak aj $\exists x F$ a $\forall x F$ sú kvantifikované formuly.

Symboly $\exists$ a $\forall$ sa nazývajú existenčný a všeobecný kvantifikátor. Pre danú kvantifikovanú formulu, premenná, ktorá sa vystupuje vo formuli, ale nie je viazaná žiadnym kvantifikátorom, sa nazýva *voľná premenná*. Ak F je kvantifikovaná formula nad X a $a \in \{0,1\} \cup X$, tak $F|_{x:=a}$ je formula ktorú získame substitúciou a za každý voľný výskyt substituovateľnej premennej x v F.

Priradenie hodnôt premenným je zobrazenie z X do $\{0,1\}$. Pre danú kvantifikovanú formulu F a priradenie hodnôt H, je *hodnota F v H* definovaná prirodzeným spôsbom.

Booleovskou funkciou n premenných nazveme každú funkciu z $\{0,1\}^n$ do $\{0,1\}$. Špeciálne pre $n = 0$ $(1,2)$ hovoríme o konštantnej (unárnej, binárnej) booleovskej funkcii. Každá formula reprezentuje nejakú booleovskú funkciu a naopak, každú booleovskú funkciu môžeme reprezentovať ako formulu. Nasledujúce pojmy nám umožňujú zapisovať formuly vo vhodných tvaroch.

Definícia 1.3 *Nech F je formula.* Literál *je booleovská premenná alebo jej negácia (t.j. x alebo $\neg x$).* Klauzula *je disjunkcia literálov (napr. $K = (x_1 \vee \neg x_2 \vee x_3)$). Formula F je v* konjunktívnom normálnom tvare*, ak je konjunkciou klauzulí, t.j. $F = K_1 \wedge K_2 \wedge \cdots \wedge K_m$.*

Platí, že ku každej formule sa dá zostrojiť s ňou ekvivalentná formula v konjunktívnom normálnom tvare.

Časť I

Algoritmy a zložitosť

Kapitola 2

Problémy a algoritmy

Algoritmus je podrobný návod (krok po kroku) na riešenie problému. Čo je problém? Na túto otázku sa pokúsime dať odpoveď tým, že ukážeme niekoľko konkrétnych problémov[1].

Rubikova kocka Pre každú konfiguráciu Rubikovej kocky sa pýtame, aký je najmenší počet krokov potrebných na to, aby sme dostali monochromatickú konfiguráciu (t.j. na každej stene sú štvorčeky rovnakej farby)? Toto je príklad *konečného problému*, pretože existuje len konečne veľa konfigurácii Rubikovej kocky. To nám dovoľuje napr. zostrojiť tabuľku, obsahujúcu riešenie každého prípadu konečného problému. Z nášho pohľadu nie sú takéto problémy zaujímavé; budeme sa zaoberať rozsiahlejšími problémami, majúcimi nekonečný počet prípadov.

Dosiahnuteľnosť v grafe Formulácia problému DOSIAHNUTEĽNOSŤ je nasledovná: je daný graf $G = (V, H)$ a dva jeho vrcholy i a j. Existuje cesta z i do j v G? Očakávanou odpoveďou je „áno“ alebo „nie“. Problémy takéhoto typu sa nazývajú *rozhodovacie* problémy. Niekedy sa používa aj označenie rozpoznávacie problémy: ak interpretujeme problém ako úlohu rozpoznať tie trojice (G, i, j), pre ktoré existuje cesta z i do j v G.

To, čo nás pri každom probléme zaujíma, je algoritmus na jeho riešenie. Z teórie vyčísliteľnosti vieme, že nie každý problém je algoritmicky riešiteľný. Pre mnohé algoritmicky riešiteľné problémy nie je ťažké nájsť algoritmus. Pre ich vyriešenie stačí preskúšať všetky do úvahy prichádzajúce možnosti (tak je to v mnohých kombinatorických a grafových problémoch). Žiaľ, takých možností býva veľmi mnoho, a preto má zmysel hľadať lepšie algoritmy. Základným kritériom, podľa ktorého posudzujeme kvalitu algoritmu, je počet elementárnych operácií, ktoré potrebujú na vyriešenie problému. V nasledujúcej kapitole definujeme Turingov stroj ako formálny model, umožňujúci presne špecifikovať ľubovoľný algoritmus. To nám umožní aj presne stanoviť kritéria na posudzovanie kvality algoritmov. Zatiaľ sa uspokojíme len s neformálnym popisom algoritmu na riešenie problému DOSIAHNUTEĽNOSŤ.

Nech máme daný graf $G = (\{1, 2, \ldots, n\}, H)$ a zaujíma nás existencia cesty z vrcholu i do vrcholu j. Počas celého výpočtu si budeme pamätať množinu S; na začiatku je $S = \{i\}$. Každý vrchol môže byť *označený* alebo *neoznačený*. To, že vrchol x je označený, indikuje, že tento vrchol už bol (alebo práve je) prvkom množiny S. V každom cykle algoritmu vyberieme vrchol x z množiny S a postupne prechádzame všetky hrany $(x, y) \in H$. Ak vrchol y nie je označený, tak ho označíme a zaradíme ho do S. Celý proces sa opakuje, až kým množina S nie je prázdna. Cesta z vrcholu i do vrcholu j existuje práve vtedy, ak na konci výpočtu je vrchol j označený. Technické otázky týkajúce sa konkrétnej realizácie ponechávame zatiaľ stranou; vrátime sa k nim neskôr.Maximálny počet hrán v grafe o n vrcholoch je n^2. V navrhnutom algoritme sa každá hrana preskúmava práve raz a preto celkovú zložitosť algoritmu možno vyjadriť ako $\mathcal{O}(n^2)$.

Z praktického hľadiska existencia algoritmu riešiaceho daný problém ešte nemusí znamenať, že daný problém vieme skutočne riešiť. To, čo potrebujeme, je *prakticky realizovateľný* (prevediteľný) algoritmus. Pod pojmom prakticky realizovateľný pritom rozumieme schop-

[1]Kvôli odlíšeniu budeme v texte písať názvy problémov veľkými písmenami

nosť nájsť riešenie daného prípadu problému v rámci daných výpočtových zdrojov. Pojem praktickej realizovateľnosti je samozrejme veľmi závislý od kontextu. Spoločnou charakteristikou je, že ak veľkosť niektorého výpočtového zdroju rastie exponenciálne v závislosti od veľkosti riešeného prípadu, tak algoritmus je použiteľný len na riešenie prípadov veľmi malého rozsahu. Sekvenčný algoritmus preto považujeme za prakticky realizovateľný, ak jeho časová náročnosť je polynomiálna, t.j. ak každý prípad problému veľkosti n vieme riešiť v čase n^k, kde k je nejaká (fixná) konštanta.

Navrhnutý algoritmus na riešenie problému DOSIAHNUTEĽNOSŤ patrí do triedy polynomiálnych algoritmov. V teórii zložitosti sa nájdenie polynomiálneho algoritmu považuje za uspokojivé riešenie problému. Náročnosť riešenia takéhoto problému sa totižto s rastúcim rozsahom (veľkosťou) prípadu zvyšuje „rozumným" spôsobom—v porovnaní napríklad s algoritmami, ktorých zložitosť je 2^n alebo dokonca $n!$. Problémy, ktoré nevieme riešiť algoritmami polynomiálnej zložitosti, považujeme za prakticky neriešiteľné (porovnaj s pojmom algoritmicky neriešiteľné—algoritmicky neriešiteľný problém je taký, ku ktorému *neexistuje* algoritmus, zatiaľ čo pre prakticky neriešiteľný problém *nepoznáme efektívny* algoritmus. Pritom môže nastať situácia, že efektívny algoritmus *neexistuje*, alebo ho *nepoznáme* (a nevieme, či existuje alebo nie)). Na riešenie takýchto náročných problémov sa používajú iné metódy, s ktorými sa stretneme v týchto učebných textoch.

Delenie problémov na prakticky riešiteľné (feasible countable) a prakticky neriešiteľné (intractable) podľa toho, či existuje alebo neexistuje polynomiálny algoritmus, samozrejme nie je úplne jednoznačné. Existujú efektívne výpočty, ktoré nie sú polynomiálne a naopak, niektoré polynomiálne algoritmy nie sú prakticky použiteľné. Ako príklad nám môže poslúžiť problém lineárneho programovania: v praxi najčastejšie (a s dobrými výsledkami) používaný simplexový algoritmus nie je polynomiálnej zložitosti, zatiaľ čo prvý známy polynomiálny algoritmus na riešenie tohoto problému (tzv. elipsoidný) nie je prakticky použiteľný. Tento rozpor vyplýva z toho, že algoritmus polynomiálnej zložitosti n^{88} je rýchlejší ako algoritmus subexponenciálnej zložitosti $n^{\log n}$ až pre problémy veľmi veľkého rozsahu, s akými sa v praxi nestretneme. V skutočnosti však väčšina známych a používaných polynomiálnych algoritmov má nízke exponenty a aj ich multiplikatívne konštanty sú rozumnej veľkosti [2].

Ďaľším diskutabilným bodom v nami zvolenom prístupe je, že pri vyhodnocovaní zložitosti algoritmu sme brali do úvahy všetky možné vstupy a za zložitosť sme brali zložitosť v najhoršom prípade. Ak skúmaný graf má $2n$ hrán, tak samozrejme náročnosť algoritmu bude len lineárna a nie kvadratická. Takisto algoritmus exponenciálnej zložitosti môže byť takýto zložitý len pre štatisticky nevýznamnú časť možných vstupov. *Očakávaná* zložitosť algoritmu nám teda môže povedať o zložitosti algoritmu viac, než jeho zložitosť v *najhoršom* prípade. Avšak k tomu, aby sme mohli určiť očakávanú zložitosť, musíme poznať rozdelenie vstupov, t.j. pravdepodobnosť, s akou sa jednotlivé prípady problému budú objavovať ako vstupy algoritmu. V (štruktúrnej) teórii zložitosti zvyčajne uprednostňujeme chápanie zložitosti ako zložitosti najhoršieho prípadu, zatiaľ čo pri analýze konkrétnych algoritmov nás viac zaujíma očakávaná zložitosť.

Maximálny tok v grafe Vstupom pre problém MAX TOK je orientovaný graf $G = (V, H)$. Predpokladajme, že v grafe existujú dva vrcholy s a t také, že do vrcholu s nevstupuje žiadna hrana a z vrcholu t nevystupuje žiadna hrana. Ďalej predpokladáme, že každej hrane (i, j) grafu je priradené číslo $c(i, j)$ — kapacita hrany (i, j). Tokom v grafe je priradenie nezáporných hodnôt $f(i, j)$ hranám grafu také, že $f(i, j) \leq c(i, j)$ a pre každý vrchol (s výnimkou vrcholov s a t) je súčet hodnôt priradených hranám vstupujúcim do vrcholu rovný súčtu hodnôt priradených hranám vystupujúcim z vrcholu. Veľkosť toku je súčet hodnôt priradených hranám vystupujúcim z vrcholu s. Problém MAX TOK je pre daný graf G zistiť, aká je maximálna veľkosť toku v G.

Tento problém sa líši od predchádzajúceho v tom, že spomedzi prípustných riešení (v našom príklade sú nimi všetky toky) hľadáme na základe daného kritéria (maximálna veľkosť) jedno najlepšie. Hovoríme o *optimalizačnom probléme*.

Optimalizačný problém sa dá vždy previesť na rozhodovací problém tak, že k problému pripojíme "hraničnú" hodnotu a pýtame sa, v akom vzťahu je riešenie optimalizačného pro-

[2] porovnaj napr. algoritmus zložitosti $5n^3$ a $10^{56}n$

blému k tejto hraničnej hodnote. Pre náš optimalizačný problém MAX TOK by sme rozhodovací problém mohli formulovať takto: problém MAX TOK(R) je pre daný graf G a číslo K rozhodnúť, či existuje tok veľkosti K alebo väčšej. Obidva problémy sú rádovo ekvivalentné v tom zmysle, že ak poznáme polynomiálny algoritmus na riešenie problému MAX TOK, tak vieme skonštruovať aj polynomiálny algoritmus pre MAX TOK(R) *a naopak*. Pri prechode od algoritmu pre MAX TOK(R) k algoritmu pre MAX TOK sa využije skutočnosť, že kapacita hrán grafu dáva horné ohraničenie na veľkosť toku, t.j. že veľkosť veľkosť maximálneho toku je číslo z nejakého intervalu $0, \ldots, M$. Potom maximálnu veľkosť toku môžeme nájsť binárnym prehľadávaním: pýtame sa, či je väčší ako $M/2$. V závislosti od odpovede zisťujeme v ďaľšom kroku, či hľadaná hodnota leží v intervale $M/4, \ldots M/2$ alebo $3M/4, \ldots, M$ atď.

Pre problém MAX TOK existuje polynomiálny algoritmus.

Fibonacciho čísla Nech $F(n)$ je n–té Fibonacciho číslo. Problémom je pre každé $n \in N$ vypočítať číslo $F(n)$. Tento problém je príkladom problému výpočtu funkcie. Problém môžeme interpretovať aj tak, že úlohou je nájsť riešenie, ktoré leží v jednoprvkovej množine prípustných riešení, a teda sa naň pozerať podobne ako na predchádzajúci príklad.

Problém obchodného cestujúceho Patrí k najznámejším a najviac skúmaným problémom teórie zložitosti. Jeho formulácia je veľmi jednoduchá: Majme daných n miest $1, \ldots, n$ a vzdialenosti d_{ij} medzi každými dvoma mestami (predpokladáme $d_{ij} = d_{ji} > 0$). Úlohou je nájsť najkratšiu cestu, ktorá začína aj končí v meste 1, prejde všetkými mestami a do žiadneho sa nedostane dvakrát. Problém nazveme POC. Zodpovedajúci rozhodovací problém POC(R) má ako vstup navyše číslo d a úlohou je zistiť, či existuje cesta daných vlastností a kratšia než d.

Algoritmus na riešenie problému POC sa dá nájsť veľmi jednoducho: pre dané mestá a vzdialenosti medzi nimi preskúšame všetky možnosti (je ich konečne mnoho!) a vyberieme tú najlepšiu. Avšak náročnosť tohoto algoritmu je úmerná $n!$ (existuje $\frac{1}{2}(n-1)!$ ciest, ktoré musíme vziať do úvahy).

Na riešenie problému POC nie je *známy* žiaden algoritmus polynomiálnej zložitosti. Metódy dynamického programovania umožňujú trochu znížiť hranicu $n!$ (za cenu nárastu potrebnej pamäte). Na druhej strane sú známe heuristické algoritmy, ktoré dávajú uspokojivé riešenie pre väčšinu 'typických' vstupov. Ak ale trváme na optimálnom riešení, nie je v súčasnoti známy žiaden efektívny postup. Pritom dôvodom určite nie je nedostatok záujmu.

Je možné, že polynomiálny algoritmus na riešenie problému POC *neexistuje*. Otázka existencie alebo neexistencie polynomiálneho algoritmu pre problém POC (a stovky ďaľších problémov), formulovaná ako otázka $\mathsf{P} \stackrel{?}{=} \mathsf{NP}$, je najdôležitejším otvoreným problémom teórie zložitosti. Všeobecne sa pritom predpokladá, že $\mathsf{P} \neq \mathsf{NP}$.

Dôležitou otázkou pri skúmaní problémov je spôsob ich prezentácie. V teórii zložitosti je obecne prijatý a používaný pohľad na problém ako na formálny jazyk. Aby sme mohli ukázať opodstatnenosť takéhoto pohľadu, musíme objasniť dve veci: ako každý problém reprezentovať reťazcami nad konečnou abecedou a prečo sa môžeme na rozhodovacie problémy a na funkcie pozerať ako na jazyky.

Existuje mnoho spôsobov, ako možno jednotlivé prípady problému reprezentovať pomocou reťazcov. Ktoré z nich sú najvhodnejšie, resp. môže výber spôsobu kódovania ovplyvniť zložitosť algoritmu na riešenie daného problému? Za *prijateľné* kódovanie budeme považovať také, ktoré má nasledovné vlastnosti:

- Kódy jednotlivých prípadov problému sa musia dať ľahko vygenerovať. Vylučujeme kódovania typu — kód „áno" prípadu začína symbolom 1 a kód „nie" prípadu začína symbolom 0. Zložitosť algoritmu by sa tak len presunula do zložitosti kódovania.

- Kódovanie má byť kompaktné, t.j. dĺžka kódu má zhruba zodpovedať veľkosti prípadu.

Za prijateľné kódovania problému budeme považovať také, ktoré sú v polynomiálnom vzťahu. To znamená, že A a B sú prijateľné kódovania tej istej množiny prípadov práve vtedy, keď v kódovaní A je daný prípad reprezentovaný reťazcom dĺžky n a v kódovaní B je reprezentovaný reťazcom dĺžky nanajvýš $p(n)$, kde p je nejaký polynóm. Špeciálnu pozornosť si zasluhuje kódovanie čísel. Zmysluplným je zrejme unárne, tak ako aj binárne alebo desiatkové kódovanie. Medzi nimi ale nie je polynomiálny vzťah (unárny kód si vyžaduje exponenciálne viac symbolov než binárny). To sa následne odrazí aj v zložitosti: stroje pracujúce s unárnym kódovaním budú mať menšiu zložitosť. Preto kedykoľvek sa v ďalšom stretneme s problémom týkajúcim sa čísel, budeme vždy predpokladať ich binárnu reprezentáciu, resp. reprezentáciu, ktorá je s ňou v polynomiálnom vzťahu.

Ukážeme ďalej, prečo sa na rozhodovací problém môžeme pozerať ako na formálny jazyk. Jazyk L asociovaný s problémom obsahuje práve tie reťazce, ktoré sú kódom „áno" prípadov problému. Príslušnosť reťazca x do jazyka L znamená, že prípad y problému, ktorého kódom je reťazec x, má kladné riešenie.

Podobnú transformáciu môžeme previesť aj pre problém výpočtu funkcie. Pri fixovanom kódovaní bude s funkciou F asociovaný jazyk $L(F) = \{\langle x, F(x)\rangle\}$, kde x je kód argumentu funkcie a $F(x)$ je kód zodpovadajúcej funkčnej hodnoty. Ak vieme vypočítať funkciu, tak vieme rozhodovať s ňou asociovaný jazyk a naopak.

Dôležitosť uvedených úvah je v tom, že poukazujú na možnosť použiť pri skúmaní problémov a ich zložitosti matematický aparát a nástroje známe z teórie formálnych jazykov a automatov.

Kapitola 3

Výpočtové modely a zložitosť

Centrálnymi pojmami teórie zložitosti sú algoritmus a zložitosť. Samozrejme pre to, aby sme ich mohli skúmať, potrebujeme ich mať presne vymedzené, definované. To vedie k definícii abstraktného matematického stroja (výpočtového modelu, výpočtového zariadenia) ako ekvivalentu pojmu algoritmus. Všeobecne, úlohou matematického stroja je transformovať daný *vstup* na zodpovedajúci *výstup*.

Historicky prvým výpočtovým modelom bol Turingov stroj, ktorý navrhol anglický matematik Alan Turing v roku 1936, teda ešte pred skonštruovaním prvého počítača. Pre svoju jednoduchosť je dodnes najpoužívanejším výpočtovým modelom v teoretických prácach. Na druhej strane práve táto jeho jednoduchosť má za následok, že je nevhodný pri konštrukcii konkrétnych algoritmov. Je to dané sčasti tým, že od reálnych počítačov sa odlišuje v niektorých dôležitých aspektoch. Hlavný rozdiel medzi Turingovymi strojmi a reálnymi počítačmi je v tom, že Turingov stroj má lineárny prístup k pamäti, zatiaľ čo reálne počítače majú priamy prístup k pamäti.

Ako výpočtový model, ktorý sa snaží preklenúť spomínaný rozdiel v prístupu k pamäti, boli navrhnuté stroje RAM. Sú vhodnejšie pre analýzu zložitosti konkrétnych algoritmov. Dôležité je pritom poznať vzťah medzi zložitostnými mierami oboch modelov—umožní nám to výsledky získané pre jeden model použiť aj pre druhý model.

Posledným výpočtovým modelom, s ktorým budeme pracovať v týchto učebných textoch sú logické obvody. Tiež majú priamy vzťah k reálnym počítačom, ovšem na tej najnižšej úrovni. Vďaka svojej jednoduchosti však zohrávajú významnú úlohu v teoretických úvahách.

3.1 Turingove stroje

Existuje viacero rôznych definícií Turingovych strojov. My budeme pracovať s nižšie popísanou variantou. V nasledujúcej kapitole ukážeme, že výber konkrétnej varianty nijako neovplyvní podstatné zložitostné charakteristiky výpočtov.

Neformálne, k–páskový Turingov stroj (skrátene TS) má k pások, ktoré sú rozdelené na jednotlivé políčka, a k hláv, z ktorých každá sa pohybuje po jednej z pások. Hlava sa môže pohybovať po páske doľava a doprava a okrem toho môže čítať a zapisovať symboly. Na každom políčku pásky je zapísaný jeden z konečne veľa páskových (pracovných) symbolov. Každá páska je jednosmerne nekonečná. Okrem k hláv a k pások má TS k dispozícii ešte konečnú množinu stavov; z nich jeden je vždy aktuálny (hovoríme, že stroj sa nachádza v danom stave).

Výpočet začína v počiatočnom stave. Každá hlava sníma prvé (najľavejšie) políčko pásky, na ktorom je zapísaný špeciálny symbol $\triangleright$, označujúci ľavý koniec pásky. Na začiatku výpočtu je na druhom až $n+1$–vom políčku prvej pásky zapísaný vstupný reťazec dĺžky n. Ostatných nekonečne veľa políčok napravo od vstupu je prázdnych—túto skutočnosť vyjadrujeme pomocou špeciálneho znaku $\sqcup$. Políčka ostatných pások napravo od $\triangleright$ sú takisto prázdne. V jednom kroku výpočtu stroj v závislosti od aktuálneho stavu a od k symbolov snímaných hlavami:

1. zmení svoj stav,

2. zapíše symbol na každé políčko snímané hlavou (čím prepíše symbol, ktorý tam bol zapísaný predtým),
3. posunie každú hlavu o jedno políčko doprava alebo doľava.

Spôsob, akým sa má zmeniť stav, prepísať obsah políčok a posunúť hlavy, predpisuje prechodová funkcia δ. Výpočet končí keď stroj prejde do jedného zo špeciálnych stavov q_{accept}, q_{reject}, q_{stop}. Na niektorých vstupoch môže výpočet bežať donekonečna; v takomto prípade hovoríme, že výpočet stroja na danom vstupe je nekonečný.

Formálna definícia Turingovho stroja

k-páskový (deterministický) Turingov stroj (pre $k \geq 1$) je 7–tica $\mathcal{M} = (Q, \Sigma, \Gamma, \triangleright, \sqcup, \delta, q_0)$. Pritom

- Q je konečná množina *stavov*; $q_0 \in Q$ je *počiatočný stav*,
- Σ je konečná *vstupná abeceda*,
- Γ je konečná *pracovná abeceda*, obsahujúca ako svoju podmnožinu vstupnú abecedu,
- $\triangleright \in \Gamma \setminus \Sigma$ je *ľavá koncová značka*,
- $\sqcup \in \Gamma \setminus \Sigma$ je symbol označujúci *prázdne políčko*
- $\delta : Q \times \Gamma^k \longrightarrow (Q \cup \{q_{stop}, q_{accept}, q_{reject}\}) \times ((\Gamma \setminus \{\sqcup\}) \times \{-1, 1\})^k$ je *prechodová funkcia*.

Predpokladáme, že stavy q_{stop}, q_{accept} a q_{reject} nie sú prvkami množiny stavov Q. Naviac požadujeme, aby Turingov stroj nikdy neprepísal ľavú koncovú značku iným symbolom a aby nikdy neposunul žiadnu zo svojich hláv vľavo od symbolu obsahujúceho ľavú koncovú značku. Formálne, požadujeme, aby v prípade, že $\delta(q, a_1, \ldots, a_k) = (p, b_1, d_1 \ldots, b_k, d_k)$ a $a_i = \triangleright$, tak $b_i = \triangleright$ a $d_i = 1$.

Konfiguácia

Výpočet Turingovho stroja môžeme definovať pomocou pojmu konfigurácia. Intuitívne, konfigurácia obsahuje úplnú informáciu o momentálnom stave výpočtu, tj. o aktuálnom stave, o polohách jednotlivých hláv a o obsahu všetkých k pások. Po konečnom počte krokov výpočtu je na každej z pások len konečne veľa políčok, ktorých obsah je neprázdny, tj. rôzny od $\sqcup$. Preto obsah každej pásky dokážeme *jednoznačne* popísať *konečným* reťazcom tvaru $y\sqcup^\omega$, kde $y \in (\Gamma \setminus \{\sqcup\})^*$ a symbol $\sqcup^\omega$ zastupuje nekonečný reťazec

$$\sqcup \sqcup \sqcup \sqcup \sqcup \sqcup \sqcup \sqcup \ldots$$

Formálne, *konfigurácia* je prvok množiny $Q \times (\{y\sqcup^\omega \mid y \in (\Gamma \setminus \{\sqcup\})^*\} \times \mathbb{N})^k$. Konfigurácia $(q, z_1, n_1, \ldots, z_k, n_k)$ špecifikuje, že stroj je v stave q, obsah i–tej pásky je z_i a i–ta hlava sníma n_i–te políčko zľava. *Počiatočná konfigurácia* stroja $\mathcal{M}$ na vstupe $w \in \Sigma^*$ je konfigurácia

$$(q_0, \triangleright w \sqcup^\omega, 1, \triangleright \sqcup^\omega, 1, \ldots, \triangleright \sqcup^\omega, 1)$$

Konfiguráciu $(q, z_1, n_1, \ldots, z_k, n_k)$ nazývame *koncovou*, ak $q \in \{q_{accept}, q_{reject}, q_{stop}\}$. Špeciálne, ak $q = q_{accept}$ $(q = q_{reject})$, tak hovoríme o *akceptujúcej* (*zamietajúcej*) konfigurácii.

Krok výpočtu a výpočet

Na množine konfigurácií definujeme reláciu *krok výpočtu*, značíme ju $\vdash_{\mathcal{M}}$ resp. $\vdash$ ak $\mathcal{M}$ je zrejmé z kontextu. Dve konfigurácie sú v relácii krok výpočtu intuitívne práve vtedy, ak v súlade s prechodovou funkciou môže stroj $\mathcal{M}$ prejsť z prvej konfigurácie do druhej.

Vo formálnej definíci relácie krok výpočtu využijeme nasledovnú notáciu. Pre ľubovoľný (aj nekonečný) reťazec z nad abecedou Γ nech $n : z$ označuje n–tý symbol reťazca z. Ďalej nech $S_b^n(z)$ označuje reťazec ktorý získame, ak n–tý symbol reťazca z nahradíme symbolom b. Potom relácia $\vdash$ je definovaná predpisom

$$(q, z_1, n_1, \ldots, z_k, n_k) \ \vdash \ (p, S_{b_1}^{n_1:z_1}(z_1), n_1 + d_1, \ldots, S_{b_k}^{n_k:z_k}(z_k), n_k + d_k)$$

práve ak

$$\delta(q, n_1 : z_1, \ldots, n_k : z_k) = (p, b_1, d_1, \ldots, b_k, d_k).$$

Výpočet TS $\mathcal{M}$ na vstupe w je postupnosť konfigurácií $K_0, \ldots K_m$ taká, že $K_i \vdash K_{i+1}$ pre $i = 0, \ldots, n-1$ a K_0 je počiatočná konfigurácia $\mathcal{M}$ na w.

Pretože sme požadovali, aby prechodová funkcia δ bola úplne špecifikovaná, jediný dôvod, pre ktorý stroj nemôže pokračovať vo výpočte je, že prejde do koncovej konfigurácie. V takomto prípade hovoríme, že výpočet stroja $\mathcal{M}$ sa *zastavil*. Naviac, ak sa výpočet zastavil v akceptujúcej konfigurácii, tak hovoríme, že $\mathcal{M}$ *akceptoval* vstupné slovo; ak výpočet zastavil v zamietajúcej konfigurácii, tak $\mathcal{M}$ vstup *zamietol*. Ak na vstupe w sa $\mathcal{M}$ zastaví, tak môžeme definovať *výstup* stroja $\mathcal{M}$ na w, označujeme ho $\mathcal{M}(w)$. Ak $\mathcal{M}$ akceptuje resp. zamieta, tak $\mathcal{M}(w) \stackrel{\text{def}}{=} ACCEPT$ resp. $\mathcal{M}(w) \stackrel{\text{def}}{=} REJECT$. V prípade, že výpočet stroja $\mathcal{M}$ sa zastavil v konfigurácii so stavom q_{stop}, tak výstupom je reťazec práve zapísaný na poslednej, k–tej páske. Pretože výpočet sa zastavil po konečnom počte krokov, pozostáva tento reťazec zo symbolu $\rhd$, za ním nasleduje konečný reťazec y, (jeho posledný symbol je rôzny od $\sqcup$) a za ním ďalší reťazec pozostávajúci zo symbolov $\sqcup$. Za výstup stroja $\mathcal{M}$ považujeme v takomto prípade reťazec y; $\mathcal{M}(w) \stackrel{\text{def}}{=} y$. Samozrejme, je možné, že výpočet stroja $\mathcal{M}$ na vstupe w sa nikdy nezastaví. Pre tento prípad definujeme $\mathcal{M}(w) \stackrel{\text{def}}{=} \nearrow$.

Rekurzivita

Turingov stroj je vhodný nástroj na riešenie rôznych typov úloh. Rozlišujeme dva základné prístupy—Turingov stroj ako *akceptor* a ako *funkcia*. Definujeme oba pojmy presne.

Definícia 3.1 (Rozhodovanie a akceptovanie) *Nech $\mathcal{M}$ je Turingov stroj so vstupnou abecedou Σ. Stroj $\mathcal{M}$* rozhoduje *jazyk $L \subseteq \Sigma^*$ práve ak pre každé slovo $w \in L$ je $\mathcal{M}(w) = ACCEPT$ a pre každé slovo $w \notin L$ je $\mathcal{M}(w) = REJECT$. Jazyk, ktorý je rozhodovaný nejakým Turingovym strojom, sa nazýva* rekurzívny *jazyk.*

Stroj $\mathcal{M}$ akceptuje *jazyk $L \subseteq \Sigma^*$ práve ak pre každé slovo $w \in L$ je $\mathcal{M}(w) = ACCEPT$ a pre každé slovo $w \notin L$ je $\mathcal{M}(w) = REJECT$ alebo $\mathcal{M}(w) = \nearrow$. Jazyk, ktorý je akceptovaný nejakým Turingovym strojom, sa nazýva* rekurzívne vyčísliteľný *jazyk.*

Jazyk rozhodovaný strojom $\mathcal{M}$ sa zvykne označovať $L(\mathcal{M})$. Medzi rozpoznávaním a akceptovaním je podstatný rozdiel. Pri rozhodovaní dostaneme v konečnom čase pre každý vstup definitívnu odpoveď $ACCEPT$ alebo $REJECT$. Pri akceptovaní dostaneme určitú odpoveď len pre vstup $w \in L$. Ak $w \notin L$, výpočet stroja môže pokračovať donekonečna. Z praktického dôvodu je táto odpoveď neuspokojivá, pretože nikdy nevieme povedať, či už sme čakali dostatočne dlho nato, aby sme mohli povedať, že slovo nepatrí do jazyka L.

Definícia 3.2 *Nech f je úplná funkcia $f : \Sigma^* \to \Sigma^*$ a nech $\mathcal{M}$ je Turingov stroj so vstupnou abecedou Σ. Hovoríme, že stroj $\mathcal{M}$* počíta *funkciu f, ak pre každý reťazec $x \in \Sigma^*$ je $\mathcal{M}(x) = f(x)$. Funkcia f počítaná nejakým Turingovym strojom sa nazýva* rekurzívna funkcia.

Príklad 3.3 *Navrhneme Turingov stroj $\mathcal{M}$ akceptujúci jazyk $L = \{a^m b^m c^m \mid m \geq 0\}$. Stroj nejprv posúva svoju hlavu až na konec vstupu a kontroluje, či reťazec zapísaný na páske je tvaru $a^*b^*c^*$. Pritom vôbec nemení obsah pásky. Keď narazí na prvé prázdne políčko, začne sa posúvať doľava až na ľavý koniec pásky. Nasleduje cyklus, v ktorom hlava prepíše symbolom X vždy jeden symbol a, jeden b, jeden c a vrátí se na ľavý koniec pásky. Keď vstupný reťazec patrí do jazyka L, tak stroj nakoniec prepíše všetky symboly a, b, c symbolom X a akceptuje. V opačnom prípade vstup zamietne.*

Nech $\mathcal{M} = (Q, \Sigma, \Gamma, \rhd, \sqcup, \delta, q_0, q_{accept}, q_{reject})$, kde

$Q = \{q_0, q_1, q_2, q_3, q_4, q_5, q_6, q_{accept}, q_{reject}\}$,

$\Sigma = \{a, b, c\}$,

$\Gamma = \Sigma \cup \{\triangleright, \sqcup, X\}$.

Prechodová funkcia je určená nasledujúcou tabuľkou:

	$\triangleright$	a	b	c	$\sqcup$	X
q_0	$(q_0, \triangleright, R)$	(q_0, a, R)	(q_1, b, R)	(q_2, c, R)	(q_3, X, L)	$-$
q_1	$-$	$(q_{reject}, -, -)$	(q_1, b, R)	(q_2, c, R)	(q_3, X, L)	$-$
q_2	$-$	$(q_{reject}, -, -)$	$(q_{reject}, -, -)$	(q_2, c, R)	(q_3, X, L)	$-$
q_3	$(q_4, \triangleright, R)$	(q_3, a, L)	(q_3, b, L)	(q_3, c, L)	(q_3, X, L)	(q_3, X, L)
q_4	$-$	(q_5, X, R)	$(q_{reject}, -, -)$	$(q_{reject}, -, -)$	$(q_{accept}, -, -)$	(q_4, X, R)
q_5	$-$	(q_5, a, R)	(q_6, X, R)	$(q_{reject}, -, -)$	$(q_{reject}, -, -)$	(q_5, X, R)
q_6	$-$	$-$	(q_6, b, R)	(q_3, X, L)	$(q_{reject}, -, -)$	(q_6, X, R)

Symbol — v tabuľke vyjadruje, že nie je podstatné, čo sa zapíše na uvedené miesto (môžeme tam doplniť čokoľvek vyhovujúce definícii). Výpočet stroja $\mathcal{M}$ na vstupe $a^2b^2c^2$ je

$$
\begin{array}{llllll}
(q_0, \triangleright aabbcc\sqcup^\omega, 1) & \vdash & (q_0, \triangleright aabbcc\sqcup^\omega, 2) & \vdash & (q_0, \triangleright aabbcc\sqcup^\omega, 3) & \vdash \\
(q_0, \triangleright aabbcc\sqcup^\omega, 4) & \vdash & (q_1, \triangleright aabbcc\sqcup^\omega, 5) & \vdash^3 & (q_2, \triangleright aabbcc\sqcup^\omega, 8) & \vdash \\
(q_3, \triangleright aabbcc\sqcup^\omega, 7) & \vdash^7 & (q_4, \triangleright aabbcc\sqcup^\omega, 2) & \vdash & (q_5, \triangleright Xabbcc\sqcup^\omega, 3) & \vdash^2 \\
(q_6, \triangleright XaXbcc\sqcup^\omega, 5) & \vdash^2 & (q_3, \triangleright XaXbXc\sqcup^\omega, 5) & \vdash^5 & (q_4, \triangleright XaXbXc\sqcup^\omega, 2) & \vdash^2 \\
(q_5, \triangleright XXXbXc\sqcup^\omega, 4) & \vdash^2 & (q_6, \triangleright XXXXXc\sqcup^\omega, 6) & \vdash^2 & (q_3, \triangleright XXXXXX\sqcup^\omega, 6) & \vdash^6 \\
(q_4, \triangleright XXXXXXX\sqcup^\omega, 2) & \vdash^7 & (q_4, \triangleright XXXXXXX\sqcup^\omega, 9) & \vdash & (q_{accept}, \triangleright XXXXXXXX\sqcup^\omega, 8) &
\end{array}
$$

Vráťme sa k úvahám, ktoré nás viedli k zavedeniu pojmu Turingov stroj. Našim pôvodným cieľom bolo nájsť model, umožňujúci presnú, matematickú formalizáciu ľubovoľného algoritmu. Prezentovaný model—Turingov stroj—je nástroj, ktorý je evidentne vhodný na riešenie problémov týkajúcich sa reťazcov a jazykov. Avšak, ako sme dokumentovali aj na našich úvodných príkladoch, problémy, s ktorými sa v praxi stretávame, sa týkajú najrozmanitejších objektov: grafov, sietí, čísel, množín atď. Ukážeme, ako možno s každým takýmto problémom asociovať vhodný jazyk.

Pretože chceme porovnávať problémy, ktoré sa objavujú v rôznych oblastiach a používajú rozdielnu abecedu, zdá sa nevyhnutným „preložiť" všetky do jednej abecedy. Takýto preklad by ovšem mal byť „šetrný" v tom zmysle, že nové slovo by nemalo byť oveľa dlhšie než pôvodné. Napríklad unárna reprezentácia čísla by bola exponenciálne dlhšia než binárna a analýza algoritmu pracujúceho s unárnymi číslami by mohla priniesť (nerealisticky) dobré výsledky. Ďalšia požiadavka je, aby preklad bol efektívne vypočítateľný: v opačnom prípade by sa zložitosť problému mohla skrývať v jeho reprezentácii. Skúsme uvážiť nasledujúci príklad problému NAJDLHŠIA CESTA. Je daný graf $G = (V, H)$, dva jeho vrcholy n_1 a n_2 a prirodzené číslo k. Úlohou je rozhodnúť, či medzi vrcholmi n_1 a n_2 existuje cesta dĺžky aspoň k. Ak prípad tohoto problému reprezentujeme ako zoznam všetkých ciest v grafe, tak ľahko nájdeme efektívny algoritmus na jeho riešenie. Stačí nájsť najdlhšiu z ciest a overiť, či jej dĺžka je aspoň k. Naopak, ak uvážime obvyklejšiu reprezentáciu grafu pomocou matice susednosti, tak nie je známy žiaden algoritmus subexponenciálnej zložitosti na hľadanie najdlhšej cesty v grafe.

Akonáhle fixujeme reprezentáciu problému, tak algoritmus na riešenie rozhodovacieho problému je Turingov stroj, ktorý rozhoduje zodpovedajúci jazyk. To znamená, že ak vstupom je kód „áno" prípadu problému, tak stroj ho akceptuje, inak (tj. ak vstupom je kód „nie" prípadu, resp. ak vstup nekóduje žiaden prípad problému) zamietne. Podobne algoritmus na riešenie optimalizačného problému je Turingov stroj, ktorý počíta funkciu (opätovne predpokladáme, že jednotlivé prípady problému, ako aj riešenia, sa vhodne reprezentujú reťazcami).

Zrejme tento pohľad je dostatočne všeobecný. Každý konečný matematický objekt môžeme reprezentovať konečným reťazcom nad vhodnou abecedou. Pre väčšinu často uvažovaných objektov je zaužívané štandartné kódovanie.

Univerzálny Turingov stroj

Medzi nami definovaným výpočtovým modelom—Turingovym strojom—a reálnym počítačom je zrejmý a podstatný jeden rozdiel. Pre výpočet každej funkcie musíme navrhnúť zvláštny Turingov stroj, zatiaľ čo pre reálny počítač postačí napísať vhodný program. Pre Turingove stroje je však tiež možný takýto prístup: je možné skonštruovať Turingov stroj, na ktorom, použijúc vhodný „program", je možné simulovať správanie ľubovoľného iného Turingovho stroja. Presnejšie, *univerzálny* Turingov stroj $\mathcal{U}$ interpretuje svoj vstup ako kód (popis) iného Turingovho stroja, nazvime ho $\mathcal{M}$, zreťazený s kódom vstupu tohto stroja, nazvime ho w [1]. Úlohou stroja $\mathcal{U}$ je simulovať výpočet stroja $\mathcal{M}$ na vstupe w. Inak vyjadrené, požadujeme aby

$$\mathcal{U}(\langle \mathcal{M}, w \rangle) = \mathcal{M}(w).$$

Veta 3.4 *Existuje univerzálny Turingov stroj.*

Dôkaz Univerzálny Turingov stroj pre vstup $\langle \mathcal{M}, w \rangle$ simuluje výpočet stroja $\mathcal{M}$ na slove w. Predpokladajme, že $\mathcal{U}$ má dve pásky. $\mathcal{U}$ používa svoju druhú pásku na zápis aktuálnej konfigurácie stroja $\mathcal{M}$. Na zápis konfigurácie používa $\mathcal{U}$ vyššie popísané kódovanie. Na začiatku výpočtu si $\mathcal{U}$ na druhú pásku zapíše počiatočnú konfiguráciu výpočtu $\mathcal{M}$ na w.

Simulácia jedného kroku výpočtu stroja $\mathcal{M}$ prebieha nasledovne: $\mathcal{U}$ číta druhú pásku až kým nenarazí na reťazec, kódujúci aktuálny stav stroja $\mathcal{M}$. Potom na prvej páske hľadá prvé pravidlo prechodovej funkcie pre aktuálny stav. Ak ho nájde, porovná symboly čítané hlavami stroja $\mathcal{M}$ so symbolmi vybraného pravidla. Ak medzi nimi nie je zhoda, hľadá ďalšie pravidlo pre aktuálny stav. Ak nastala zhoda, aplikuje nájdené pravidlo: patričným spôsobom upraví aktuálnu konfiguráciu. Stroj $\mathcal{U}$ ukončí svoj výpočet vtedy, keď simulovaný výpočet stroja $\mathcal{M}$ sa zastavil, pričom $\mathcal{U}(\langle \mathcal{M}, w \rangle) = \mathcal{M}(w)$.

Samozrejme môže nastať situácia, keď vstupný reťazec nie je kódom žiadneho Turingovho stroja. $\mathcal{U}$ je schopný takúto situáciu rozpoznať. Jeho výpočet na takomto vstupe bude spočívať v tom, že prejde do špeciálneho stavu, v ktorom bude stále posúvať hlavy doprava (t.j. nekonečný výpočet). □

3.1.1 Zložitostné miery

Začiatkom 60–tych rokov bola hranica medzi algoritmicky riešiteľnými a algoritmicky neriešiteľnými problémami exaktne definovaná a boli známe techniky umožňujúce pre konkrétne problémy rozhodovať, či sú alebo nie sú algoritmicky riešiteľné. Stále intenzívnejšie a širšie používanie počítačov v praxi však prinieslo so sebou otázky ako: „Ako merať efektívnosť algoritmov?", „Ako navzájom porovnať efektivitu dvoch rôznych algoritmov?", „Ako vyjadriť výpočtovú obtiažnosť problému?". Snaha nájsť odpoveď na tieto a podobné otázky viedla k formulovaniu dvoch fundamentálnych zložitostných mier, časovej a priestorovej, v prácach Hartmanisa at.al. a nezávisle v prácach Trachtenbrota. Jedná sa o *dynamické* zložitostné miery, ktoré zohľadňujú ako algoritmus, tak aj vstup a odrážajú priebeh výpočtu [2]. Obe zložitostné miery chápeme ako funkcie vstupu. Neformálne, časová zložitosť algoritmu pre daný vstup je počet elementárnych operácií vykonaných pri spracovaní uvažovaného vstupu. Priestorová zložitosť algoritmu je počet „elementárnych" pamäťových miest, použitých v priebehu výpočtu.

Čas

Definícia 3.5 (Časová zložitosť výpočtu) *Časová zložitosť výpočtu Turingovho stroja $\mathcal{M}$ na vstupe w je funkcia $t_{\mathcal{M}} : \Sigma^* \to \mathbb{N}$ definovaná predpisom*

$$t_{\mathcal{M}}(w) \stackrel{def}{=} d$$

[1] Predpokladáme, že každý Turingov stroj, ako aj jeho vstup, kódujeme pomocou binárnej abecedy $\{0, 1\}$. Spôsob kódovania, spolu s párovacou funkciou $\langle\rangle$, je presne popísaný v kapitole 1. Pre stroj (slovo) a pre jeho kód používame totožné označenie; presný význam je vždy zrejmý z kontextu.

[2] Opakom sú *statické* zložitostné miery. Typickým príkladom je zložitostná miera veľkosť programu: na každý algoritmus sa môžeme pozerať ako na slovo nad nejakou konečnou abecedou. Otázkou je, aké dlhé je toto slovo? Týmto spôsobom môžeme priradiť problému zložitosť ako dĺžku najkratšieho algoritmu, ktorý ho rieši.

ak výpočet $\mathcal{M}$ na w sa zastavil po d krokoch a

$$t_{\mathcal{M}}(w) \stackrel{def}{=} \infty$$

ak výpočet $\mathcal{M}$ na w je nekonečný.

Časová zložitosť výpočtu je teda počet krokov, ktoré stroj vykoná pred tým, než sa zastaví. Pre nás je však zaujímavé poznať nielen časovú zložitosť jednotlivých, izolovaných výpočtov. To, čo je pre nás dôležité, je poznať náročnosť riešenia ľubovoľného prípadu problému. To vedie k definícii zložitosti ako funkcii veľkosti (rozsahu) riešeného prípadu problému.

Definícia 3.6 (Časová zložitosť Turingovho stroja) *Časová zložitosť Turingovho stroja $\mathcal{M}$ je funkcia $T_{\mathcal{M}} : \mathbb{N} \to \mathbb{N} \cup \{\infty\}$ definovaná predpisom*

$$T_{\mathcal{M}}(n) \stackrel{def}{=} \max \{ t_{\mathcal{M}}(w) \mid w \in \Sigma^n \} .$$

Teraz už môžeme definivať pojem zložitostnej triedy. Zložitostná trieda je množina jazykov, ktoré zdieľajú spoločnú vlastnosť: sú rozhodované so zložitosťou (a nemusí to byť len časová zložitosť) ohraničenou danou funkciou.

Definícia 3.7 *Pre funkciu $f : \mathbb{N} \to \mathbb{N}$ definujeme*

$$\mathsf{DTIME}(f(n)) \stackrel{def}{=} \{ L \mid \begin{array}{l} \textit{existuje TS } \mathcal{M} \textit{ rozhodujúci } L \\ \textit{s časovou zožitosťou } T_{\mathcal{M}}(n) \leq f(n) \} \end{array}$$

Zložitostné triedy, vzťah medzi nimi a problémami, ktoré obsahujú, sú hlavným predmetom skúmania štruktúrnej teórie zložitosti.

Príklad 3.8 *Analyzujme časovú zložitosť stroja z príkladu 3.3, ktorý rozhoduje, či vstupný reťazec je tvaru $a^m b^m c^m$. V prvej fáze stroj číta celý vstup zľava doprava a kontroluje, či reťazec je tvaru $a^* b^* c^*$. Ak áno, hlava sa posunie na začiatok pásky a začína cyklus, v ktorom sa prepíše jeden symbol a, jeden b a jeden c. Zrejme ak slovo patrí do jazyka L, tak tento cyklus sa zopakuje presne m krát. Nakoniec stroj prechádza celú pásku a kontroluje, či všetky políčka boli označené. Celkove, počet krokov, ktoré vykoná stroj vo výpočte na vstupe dĺžky n, môžeme zhora odhadnúť číslom $2(n+1) + \frac{n}{3}(n+1) + n + 1$. Časová zložitosť stroja $\mathcal{M}$ je $\mathcal{O}(n^2)$.*

Ak časová zložitosť výpočtu stroja $\mathcal{M}$ na vstupe w je menšia ako $|w|$, tak stroj rozhodol o výstupe $\mathcal{M}(w)$ bez toho, aby poznal *celý* vstupný reťazec. V takomto prípade stroj $\mathcal{M}$ buď rozpoznáva triviálny jazyk, alebo bolo pri kódovaní asociovaného problému použité „nešetrné" kódovanie, kde istá časť kódu nenesie žiadnu informáciu. Preto sa sústredíme len na stroje, ktorých časová zložitosť na vstupe dĺžky n je aspoň $n+1$.

Priestor

Čas výpočtu Turingovho stroja samozrejme nie je jediným kritériom charakterizujúcim náročnosť prevádzaného výpočtu. Preto definujeme priestorovú zložitosť ako zložitostnú mieru odrážajúcu pamäťovú náročnosť výpočtov. Pre Turingove stroje sa zdá prirodzené definovať túto mieru ako počet políčok, ktoré na konci výpočtu obsahujú neprázdny symbol (pripomíname, že stroj nemôže zapisovať symbol $\sqcup$). Tým by sme však do priestorovej zložitosti zahrnuli aj políčka, ktoré na začiatku obsahovali vstup. Dôsledkom by bolo, že by sme neboli schopní rozlíšiť výpočty, ktorých priestorová zložitosť je menšia než dĺžka vstupu. Preto definujeme k–páskový (pre $k \geq 2$) *Turingov stroj so vstupnou páskou*. Od všeobecného k–páskový modelu sa odlišuje iba v tom, že obsah prvej pásky, na ktorej je zapísané vstupné slovo, sa počas výpočtu nesmie meniť. Na jeho označenie sa používa aj pojem *on–line* stroj, zatiaľ čo pre všeobecný model sa používa pomenovanie *off–line* stroj.

Podobný problém vznikne, keď uvažujeme Turingov stroj, ktorý počíta funkciu. Zrejme počet použitých políčok vo výpočte na vstupe w bude aspoň $|\mathcal{M}(w)|$. Tým opäť strácame možnosť rozlíšiť výpočty, v ktorých na samotný výpočet bolo treba použiť menej než $|\mathcal{M}(w)|$

políčok. Aby sme sa podobným problémom vyhli, definujeme k–páskový (pre $k \geq 3$) Turingov stroj *so vstupnou a výstupnou páskou*. Je to stroj so vstupnou páskou a dodatočným obmedzením týkajúcim sa spôsobu práce s k–tou páskou: k–ta hlava sa nikdy nemôže posunúť smerom doľava. Inými slovami, stroj môže na poslednú pásku len zapisovať, ale nikdy nemôže prečítať, resp. zmeniť už zapísaný reťazec.

Definícia 3.9 (Priestorová zložitosť výpočtu) *Nech $\mathcal{M}$ je Turingov stroj a K jeho konfigurácia. Potom priestorová zložitosť konfigurácie K, značíme $s_{\mathcal{M}}(K)$, je počet políčok, ktorých obsah je neprázdny (tj. rôzny od $\sqcup$). V prípade, že $\mathcal{M}$ je stroj so vstupnou páskou, tak do celkovej sumy nezapočítavame políčka prvej pásky. Ak $\mathcal{M}$ je stroj so vstupnou a vstupnou páskou, tak do výslednej sumy nezapočítavame ani políčka poslednej pásky.*

Priestorová zložitosť výpočtu *stroja $\mathcal{M}$ na vstupe w je funkcia* $s_{\mathcal{M}} : \Sigma^* \to (\mathbb{N} \cup \{\infty\})$ *definovaná predpisom*

$$s_{\mathcal{M}}(w) \stackrel{\text{def}}{=} \max\left\{\, s(K) \mid K \textit{ je konfigurácia výpočtu } \mathcal{M} \textit{ na } w \,\right\}.$$

Definícia 3.10 (Priestorová zložitosť Turingovho stroja) *Priestorová zložitosť Turingovho stroja $\mathcal{M}$ je funkcia $S_{\mathcal{M}} : \mathbb{N} \to \mathbb{N} \cup \{\infty\}$ definovaná predpisom*

$$S_{\mathcal{M}}(n) \stackrel{\text{def}}{=} \max\{\, s_{\mathcal{M}}(w) \mid w \in \Sigma^n \,\}.$$

Podobne ako pre čas, môžeme definovať aj zložitostné triedy pre priestor. Pre funkciu $f : \mathbb{N} \to \mathbb{N}$ definujeme

$$\mathsf{DSPACE}(f(n)) \stackrel{\text{def}}{=} \{\, L \mid \begin{array}{l}\text{existuje TS } \mathcal{M} \text{ rozhodujúci } L\\ \text{s priestorovou zožitosťou } S_{\mathcal{M}}(n) \leq f(n)\,\}\end{array}$$

Pre úplnosť uvádzame, že teória zložitosti skúma aj iné zložitostné miery Turingových strojov. Čas a priestor sú však bezosporu v centre pozornosti.

3.2 RAM

Doteraz sme skúmali Turingove stroje ako formalizáciu pojmu algoritmus. To však nie je jediný formálny model umožňujúci prezentovať algoritmy. Turingove stroje sú vhodným nástrojom na skúmanie fundamentálnych vlastností výpočtov. Na druhej strane je však architektúra Turingovych strojov veľmi vzdialená architektúre moderných počítačov. V tejto časti si ukážeme iný model, stroje RAM, ktoré sú oveľa bližšie reálnym počítačom. Metodológie návrhu algoritmov pre stroje RAM a pre reálne počítače sú veľmi podobné. Preto zložitostná analýza algoritmov pre RAM dobre popisuje aj zložitosť programov pre sekvenčné počítače. Zároveň však, ako ukážeme na záver tejto kapitoly, existuje efektívna simulácia medzi strojmi RAM a Turingovymi strojmi. Ako dôsledok tejto simulácie získame presný vzťah medzi zložitosťou výpočtov na TS a nan RAMoch.

Stroj RAM je výpočtové zariadenie, ktoré podobne ako Turingov stroj, pozostáva z programu, ktorý sa realizuje na danej dátovej štruktúre. Dátovou štruktúrou stroja RAM je pole registrov, v ktorých sa uchovávajú čísla. Hlavný rozdiel spočíva v tom, že stroje RAM môžu priamo pristupovať ku každému svojmu pamäťovému registru (čítať aj zapisovať). Táto výhoda so sebou ovšem prináša aj isté dôsledky: aby sme boli schopní pristupovať priamo ku každému registru, musíme ich vedieť adresovať; adresy musíme uchovávať v ďaľších registroch. Keďže počet registrov nie je obmedzený, nie je ohraničená ani veľkosť adries, a preto musíme pripustiť, aby sa v registroch mohli uchovávať ľubovoľne veľké čísla. Obsahy registrov potrebujeme počas výpočtu meniť, preto aj operácie stroja RAM musia byť nad ľubovoľne veľkými číslami.

Formálne, RAM program $\Pi = (\pi_1, \pi_2, \ldots, \pi_m)$ je konečná postupnosť *inštrukcií*, kde každá inštrukcia π_i je niektorého typu ako ukazuje tabuľka 3.1.

V každom kroku výpočtu stroja RAM obsahuje Register i, pre $i \geq 0$, celé číslo, ktoré označujeme r_i. Ďalej má RAM počítadlo inštrukcií P, ktorého iniciálna hodnota je 1. Jeden krok výpočtu spočíva vo vykonaní inštrukcie, ktorej číslo je uložené v počítadle P. Výsledkom každej

Inštrukcia	**Operand**	**Sémantika**
`READ`	j	$r_0 := i_j$
`READ`	$\uparrow j$	$r_0 := i_{r_j}$
`STORE`	j	$r_j := r_0$
`STORE`	$\uparrow j$	$r_{r_j} := r_0$
`LOAD`	x	$r_0 := x$
`ADD`	x	$r_0 := r_0 + x$
`SUB`	x	$r_0 := r_0 - x$
`JUMP`	j	$P := j$
`JPOS`	j	**if** $r_0 > 0$ **then** $P := j$
`JZERO`	j	**if** $r_0 = 0$ **then** $P := j$
`JNEG`	j	**if** $r_0 < 0$ **then** $P := j$
`HALT`	j	$P := 0$

j je celé číslo
r_j je obsah Registra j
i_j je j–ty vstup
x je ľubovoľný operand typu j, $\uparrow j$, $= j$
Hodnota operanda j je r_j, operanda $\uparrow j$ je r_{r_j} a hodnotou operanda $= j$ je j.
P je počítadlo inštrukcií; každá inštrukcia zvýši jeho hodnotu o 1; ak to nie je explicitne určené inak.

Tabulka 3.1: Inštrukcie stroja RAM

vykonanej inštruckie je zmena hodnoty jedného registra a zmena hodnoty počítadla P. Zmeny, t.j. sémantika inštrukcií sú definované prirodzeným spôsobom, ako sa s nimi stretávame pri reálnych počítačoch. Špeciálnu úlohu zohráva Register 0: slúži ako *akumulátor*, kde sa uskutočňujú všetky aritmetické aj logické operácie. Všimnime si tiež tri *spôsoby* (módy) *adresovania*. Niektoré inštrukcie majú *operand* x, kde x môže byť j, $\uparrow j$, alebo $= j$. Ak x je číslo j, tak jeho hodnotou je obsah Registra j, hodnotou operanda $\uparrow j$ je obsah registra, ktorého číslo je uložené v Registri j. Nakoniec ak x je $= j$, tak jeho hodnotou je samotné číslo j. Každá inštrukcia mení aj hodnotu počítadla P. S výnimkou posledných piatich inštrukcií je nová hodnotou počítadla $P + 1$. Skokové inštrukcie menia obsah P inak, v závislosti od obsahu akumulátoru. Nakoniec, inštrukcia `HALT` spôsobí ukončenie výpočtu. Každú sémanticky nesprávnu inštrukciu (napr. adresovanie registra so záporným číslom) považujeme tiež za `HALT` inštrukciu. Na začiatku je obsah všetkých registrov 0 a obsah počítadla P je 1.

Vstupom stroja RAM je konečná postupnosť celých čísel, ktoré sú obsiahnuté v konečnom poli *vstupných registrov*. Každé číslo zo vstupu môže byť presunuté do akumulátora `READ` inštrukciou. Po zastavení výpočtu je výsledok uložený v akumulátore resp. ak výsledkom je postupnosť čísiel, tak v akumulátore je uložené číslo n a v prvom až n–tom regristri sú postupne uložené výsledné hodnoty.

Nech Π je RAM program, nech D je množina konečných postupností celých čísel a nech ϕ je funkcia $\phi : D \to D$. Potom hovoríme, že Π počíta ϕ práve ak pre ľubovoľné $I \in D$ výpočet stroja RAM s programom Π a vstupom I sa zastaví s výsledkom $\phi(I)$.

Príklad 3.11 *Tabuľka 3.2 obsahuje veľmi jednoduchý RAM program na násobenie dvoch čísel, t.j. program počíta funkciu $\phi : \mathbb{N}^2 \to \mathbb{N}$, kde $\phi(i_1, i_2) = i_1 \cdot i_2$. Program i_2–krát opakuje cyklus inštrukcií 6 až 12. Na začiatku k–tej iterácie (začína sa s iteráciou 0), obsahuje Register 3 číslo $k \cdot i_1$. Na konci každej iterácie sa testuje, či Register 2 obsahuje 0; ak áno, výsledok je uložený v Registri 3; v opačnom prípade sa pokračuje ďalšou iteráciou.*

Definujeme ďalej zložitostné miery stroja RAM úplne analogickým spôsobom ako pre TS. Prirodzeným sa zdá definovať časovú zložitosť výpočtu ako počet vykonaných inštrukcií (tzv. *jednotková cena* alebo tiež *uniformná cena*). Ovšem vzhľadom na to, že každá inštrukcia môže narábať s ľubovoľne veľkým číslom, nie vždy táto miera odráža skutočnú náročnosť výpočtu a môže viesť k zavádzajúcim výsledkom. Napríklad pri vektorových operáciach môžeme celý

1.	`READ 1`
2.	`JZERO 13`
3.	`STORE 1`
4.	`READ 2`
5.	`JZERO 13`
6.	`STORE 2`
7.	`LOAD 3`
8.	`ADD 1`
9.	`STORE 3`
10.	`LOAD 2`
11.	`SUB =1`
12.	`JPOS 6`
13.	`LOAD 3`
14.	`HALT`

Tabulka 3.2: RAM program na násobenie

vektor čísel reprezentovať ako jedno (dlhé) číslo a previesť naraz operáciu nad všetkými hodnotami. Druhou možnosťou, ako definovať časovú zložitosť je brať do úvahy nielen počet inštrukcií, ale aj veľkosť čísel, s ktorými sa pri nich narába (tzv. *logaritmická cena*). Pre každé celé číslo i definujeme jeho dĺžku $l(i)$ ako dĺžku jeho binárnej reprezentácie[3], $l(i) \stackrel{\text{def}}{=} |\,bin(i)\,|$. Logaritmická cena inštrukcie je $\max\{\, l(x) \mid x$ je číslo, s ktorým inštrukcia manipuluje$\}$ (tzn. buď číslo registra, obsah registra alebo konštanta).

Pri definovaní priestorovej zložitosti takisto môžeme vychádzať buď z jednotkovej alebo z logaritmickej ceny registra. Pri prvom spôsobe je priestorová zložitosť rovná počtu registrov, ktoré boli v priebehu výpočtu použité. Pre definovanie logaritmickej zložitosti berieme do úvahy nielen počet použitých registrov, ale aj veľkosť čísel, ktoré v nich boli v priebehu výpočtu uložené. Logaritmická cena registra R je maximálna dĺžka čísla, ktoré bolo v priebehu výpočtu uložené v R. Logaritmická priestorová zložitosť výpočtu je súčet logaritmických cien všetkých registrov, použitých v priebehu výpočtu.

Zostáva nám ešte upresniť, čo rozumieme pod velkosťou vstupu stroja RAM. Nech vstupom pre RAM je postupnosť čísel $I = (i_1, \ldots, i_k)$. Pod binárnou reprezentáciou postupnosti I rozumieme reťazec $bin(I) \stackrel{\text{def}}{=} \langle bin(i_1), \ldots, bin(i_k)\rangle$. Dĺžku vstupu I definujeme ako $l(I) \stackrel{\text{def}}{=} |\,bin(I)\,|$.

Príklad 3.12 *Vyhodnoťme zložitosť stroja RAM z predchádzajúceho príkladu. Jeho časová zložitosť je určovaná počtom vykonaných iterácií, čo je práve i_2. Preto časová zložitosť výpočtu na vstupe dĺžky n pri jednotkovej cene inštrukcie je $\mathcal{O}(2^n)$, pretože $i_2 \le 2^n$. Pri logaritmickej cene inštrukcie je zložitosť $\mathcal{O}(2^n \log(2^{2n}))$, pretože najväčšie číslo, s ktorým sa v programe narába je $i_1 \cdot i_2 \le 2^{2n}$. Podobne logaritmická priestorová zložitosť nepresiahne $\mathcal{O}(n)$.*

Stroje RAM sa zdajú byť oveľa prirodzenejším výpočtovým modelom než TS, a teda aj výsledky týkajúce sa zložitosti výpočtov na RAMoch by mali byť „realistickejšie“ v porovnaní so zložitosťou výpočtov na TS. Na otázku, či tomu tak skutočne je, dávajú odpoveď nasledujúce dve simulácie, ktoré dokazujú, že obidva uvažované výpočtové modely sú *polynomiálne ekvivalentné*. To znamená, že trieda problémov, riešiteľných v polynomiálnom čase Turingovymi strojmi a strojmi RAM sú totožné. A pretože práve problémy riešiteľné v polynomiálnom čase považujeme za prakticky riešiteľné, zostáva pojem praktickej riešiteľnosti invariantný vzhľadom k zvolenému výpočtovému modelu. Samozrejme, pre naše (teoretické) úvahy je vhodnejší jednoduchší model, tj. TS. V ďalšom preto budeme pracovať predovšetkým s ním.

V nasledujúcej časti navrhneme efektívnu vzájomnú simuláciu RAM–ov a Turingovych strojov. Uvažujme jednopáskový Turingov stroj so vstupnou abecedou $\Sigma = \{a_1, \ldots, a_k\}$. Položme D_Σ rovné množine postupností čísel

$$D_\Sigma \stackrel{\text{def}}{=} \left\{\, (i_1, \ldots, i_n, 0) \mid 0 \le n,\ 1 \le i_j \le k \text{ pre } j = 1, \ldots, n \,\right\}.$$

[3] Definícia binárnej reprezentácie celého čísla je uvedená v kapitole 1.

Pre jazyk $L \subseteq \Sigma^*$ definujeme funkciu $\phi_L : D_\Sigma \to \{0,1\}$ predpisom

$$\phi_L(i_1,\dots,i_n,0) = \begin{cases} 1 & \text{ak } a_{i_1}\dots a_{i_n} \in L \\ 0 & \text{inak} \end{cases}$$

Posledná nula je v postupnosti čísel z technických dôvodov; v nasledujúcej konštrukcii umožní stroju RAM rozpoznať koniec vstupu.

Veta 3.13 *Nech L je jazyk z triedy* $\mathsf{DTIME}(f(n))$*. Potom existuje RAM program, ktorý počíta ϕ_L a jeho uniformná (logaritmická) časová zložitosť je $\mathcal{O}(f(n))$ ($\mathcal{O}(f(n)\log f(n))$).*

Dôkaz Nech $\mathcal{M} = (Q,\Sigma,\Gamma,\rhd,\sqcup,\delta,q_0)$ je Turingov stroj rozhodujúci jazyk L v čase $f(n)$. Výpočet RAM stroja, simulujúceho $\mathcal{M}$, štartuje tak, že sa okopíruje vstup do Registrov 2 až $n+1$. Do Registra 1 sa uloží číslo 1, v priebehu simulácie si v tomto registri RAM pamätá, ktoré políčko sníma hlava simulovaného Turingovho stroja.

Od tohoto okamihu program simuluje výpočet Turingovho stroja krok po kroku. Program má pre každý stav $q \in Q$ sadu inštrukcií. Každá sada obsahuje $s = |\Gamma|$ podpostupností. j–ta podpostupnosť q–tej sady začína inštrukciou, povedzme, $N_{q,j}$. Predpokladajme, že $\delta(q,a_j) = (p,a_l,D)$, kde a_x označuje x–tý symbol abecedy Γ. Inštrukcie j–tej podpostupnosti sú v tabuľke 3.3 Konštanta d má hodnotu -1, resp. 1, podľa toho, či predpísaný smer pohybu

$N_{q,j}$	`LOAD` $\uparrow 1$
$N_{q,j}+1$	`SUB` $=j$
$N_{q,j}+2$	`JZERO` $N_{q,j}+4$
$N_{q,j}+3$	`JUMP` $N_{q,j+1}$
$N_{q,j}+4$	`LOAD` $=l$
$N_{q,j}+5$	`STORE` $\uparrow 1$
$N_{q,j}+6$	`LOAD 1`
$N_{q,j}+7$	`ADD` $=d$
$N_{q,j}+8$	`STORE 1`
$N_{q,j}+9$	`JUMP` $N_{p,1}$

Tabulka 3.3: Fragment RAM–programu simulujúceho Turingov stroj

hlavy D je -1, resp. 1.

Nakoniec, stavom q_{accept} a q_{reject} zodpovedajú jednoduché postupnosti inštrukcií, ktoré uložia do akumulátoru konštantu 1 resp. 0 a ukončia výpočet RAM programu. Simulácia jedného kroku výpočtu Turingovho stroja si vyžaduje nanajvýš C krokov stroja RAM, kde C je konštanta závisiaca len na mohutnosti abecedy Γ a nezávisiaca od dĺžky vstupu. Preto časová zložitosť navrhnutého stroja RAM pri jednotkovej cene inštrukcie je $\mathcal{O}(f(n))$. Maximálne číslo, s ktorým RAM počas manipuluje, je číslo kódujúce polohu hlavy TS a jeho hodnota nepresiahne $f(n)$. preto logaritmická časová zložitosť je $\mathcal{O}(f(n)\log f(n))$. □

Nasledújúci výsledok je trochu viac prekvapivý, pretože z neho vyplýva, že každý RAM program sa dá simulovať Turingovym strojom len s polynomiálnou stratou efektívnosti. Zavedieme najprv vstupno–výstupnú konvenciu, ktorá umožní, aby sa Turingov stroj správal ako RAM program. Nech $\phi : D \to D$ je funkcia, ktorú počíta RAM. Potom Turingov stroj $\mathcal{M}$ simuluje RAM, tj. počíta funkciu ϕ, práve ak pre každé $I \in D$ platí $\mathcal{M}(bin(I)) = bin(\phi(I))$.

Veta 3.14 *Nech RAM program Π počíta funkciu $\phi : D \to D$ a jeho jednotková časová zložitosť je $f(n)$. Potom existuje 7–páskový Turingov stroj $\mathcal{M}$, ktorý počíta ϕ, pričom jeho časová zložitosť je $\mathcal{O}(f^2(n)(f(n)+n))$.*

Dôkaz Navrhneme Turingov stroj $\mathcal{M}$ počítajúci funkciu ϕ. Prvá páska stroja $\mathcal{M}$ obsahuje vstup, nikdy ju neprepisujeme. Na druhej páske je zaznamenaný okamžitý stav výpočtu stroja RAM; t.j. obsahy všetkých registrov. Zapisujeme ich ako postupnosť binárnych reťazcov $bin(i)\bot bin(r_i)$, navzájom oddelených symbolom $\bot$. Postupnosť je ukončená špeciálnym znakom, povedzme $\dashv$. Pri každej aktualizácii hodnoty Registra i sa dvojica $bin(i)\bot bin(r_i)$ prepíše (napr. znakmi □) a dvojica kódujúca nový obsah Registra i sa pripojí na koniec pásky; symbol ⊣ sa posunie vpravo. Na tretej páske je uložená aktuálna hodnota počítadla inštrukcií P. Štvrtá páska sa používa na uchovanie čísla registra, ktorého obsah potrebuje TS poznať.

Stavy stroja $\mathcal{M}$ sú rozdelené do m skupín, kde m je počet inštrukcií simulovaného RAM programu Π. Každá skupina realizuje jednu inštrukciu. Potrebné obsahy registrov zistí $\mathcal{M}$ prejdením druhej pásky zľava doprava; postačia mu nanajvýš dve prejdenia (v prípade nepriameho adresovania). Zakaždým, keď sa číta dvojica $bin(i)\perp bin(r_i)$, porovná sa obsah štvrtej pásky s reťazcom $bin(i)$. Ak sú identické, tak reťazec $bin(r_i)$ sa prepíše na štvrtú pásku. Všetky inštrukcie sa realizujú na piatej a šiestej páske. Po zrealizovaní inštrukcie sa aktualizuje obsah druhej pásky vyššie popísaným spôsobom a začína simulácia ďalšej inštrukcie. Ak simulovanou inštrukciou je inštrukcia `HALT`, tak $\mathcal{M}$ zistí na druhej páske obsah akumulátoru a výsledok zapíše na siedmu (výstupnú) pásku.

Odhad časovej zložitosti stroja $\mathcal{M}$ je založený na nasledovnom fakte.

Lema 3.15 *Po t krokoch výpočtu RAM programu Π na vstupe I má obsah každého registra dĺžku nanajvýš*

$$t + l(I) + l(B),$$

kde B je najväčšie číslo explicitne uvedené v programe Π.

Dôkaz Indukciou vzhľadom na t. Tvrdenie platí na začiatku výpočtu, pre $t = 0$. Predpokladajme, že je splnené aj po $t-1$ krokoch. Ukážeme, že platí aj po t–tom kroku.

Rozlíšime niekoľko prípadov podľa toho, aká inštrukcia sa v t–tom kroku vykonáva. Pri skokovej inštrukcii a pri inštrukcii `HALT` sa nemení obsah žiadneho registra. Inštrukcie `STORE` a `LOAD` modifikujú obsah jedného registra. Avšak novou hodnotou sa stane číslo, ktoré existovalo už v predchádzajúcom kroku výpočtu a platí preň indukčný predpoklad. Výsledkom inštrukcie `READ` je presun čísla zo vstupu do akumulátoru; veľkosť načítaného čísla zaručene nepresiahne $l(I)$. Nakoniec zostáva uvážiť aritmetické operácie. Pri sčitovaní dvoch čísiel i a j boli tieto čísla uložené v niektorom z registrov už v predchádzajúcom kroku výpočtu (a platí pre ne indukčný predpoklad), alebo sú explicitne uvedené v programe (a ich dĺžka nepresiahne $l(B)$). Dĺžka výsledku je nanajvýš jedna plus dĺžka dlhšieho z operadov i, j. Analogicky pre inštruckiu `SUB`. □

Zostáva nám ukázať, že na simuláciu jednej inštrukcie stroja RAM stačí Turingovmu stroju $\mathcal{M}$ $\mathcal{O}(f(n)^2)$ krokov, kde n je dĺžka vstupu. Dekódovanie simulovanej inštrukcie si vyžaduje konštantný počet krokov. Nájdenie hodnoty registra si vyžaduje nanajvýš $\mathcal{O}(f(n)(f(n)+n))$ krokov, pretože na druhej páske je zapísaných $\mathcal{O}(f(n))$ dvojíc; každá dĺžky nanajvýš $\mathcal{O}(f(n)+n)$ (vyplýva z uvedenej lemy). Samotná simulácia inštrukcie zahŕňa jednoduché aritmetické operácie na reťazcoch dĺžky $\mathcal{O}(f(n)+n)$, a teda si vyžaduje tiež $\mathcal{O}(f(n)+n)$ krokov. □

Poznamenávame, že ak simulovaný RAM má logaritmickú časovú zložitosť $f(n)$, tak $f(n) \geq n$ a zložitosť Turingovho stroja $\mathcal{M}$ je $\mathcal{O}(f^3(n))$[4]. Ak by sme sadu inštrukcií stroja RAM rozšírili o inštrukciu násobenia, tak tvrdenie uvedené v leme 3.15 by nebolo pravdivé. V takomto prípade by polynomiálna ekvivalencia platila len pre model RAM s logaritmickou cenou inštrukcie.

3.3 Nedeterministické Turingove stroje

Posledný z prezentovaných modelov sa od doteraz uvažovaných podstatne odlišuje v tom, že ho nemožno považovať za realistický výpočtový model. Nedeterministické Turingove stroje neslúžia na modelovanie žiadneho reálneho výpočtového zariadenia; neskôr uvidíme, že sú nástrojom vhodným skôr na formulovanie istých problémov než na ich riešenie. Ukážeme, že síce je možné ich simulovať každým z predchádzajúcich modelov, ovšem za cenu exponenciálneho nárastu časovej zložitosti. Otázka, či existuje efektívnejšia simulácia, je základným problémom teórie zložitosti a neskôr ju presne sformulujeme ako $\mathsf{P} \stackrel{?}{=} \mathsf{NP}$ problém.

[4]Nerovnosť $f(n) \geq n$ platí za predpokladu, že stroj RAM prečíta *všetky* vstupné údaje. Podobná úvaha nás viedla k predpokladu, že časová zložitost TS je aspoň n.

Nedeterministický Turingov stroj je zovšeobecnením deterministického stroja v tom, že nie je jednoznačne určený krok výpočtu; stroj si môže vybrať z niekoľkých možností. Prechodová funkcia sa zmení na reláciu a zodpovedajúcim spôsobom sa zmení aj kritérium, na základe ktorého rozhodujeme, či vstup bol prijatý alebo zamietnutý.

Definícia 3.16 (Nedeterministický Turingov stroj) *k-páskový nedetrministický Turingov stroj (pre $k \geq 1$) je 7-tica $\mathcal{M} = (Q, \Sigma, \Gamma, \triangleright, \sqcup, \delta, q_0)$. Pritom Q, Σ, Γ, $\triangleright$, $\sqcup$ a q_0 majú ten istý význam ako v definícii deterministických strojov. δ je relácia,*

$$\delta \subseteq (Q \times \Gamma^k) \times [(Q \cup \{q_{accept}, q_{reject}, q_{stop}\}) \times ((\Gamma - \{\sqcup\}) \times \{-1, 1\})^k].$$

Pojem konfigurácie nedeterministického stroja je totožný s pojmom konfigurácie deterministického stroja. Podobne pre reláciu krok výpočtu. Na rozdiel od deterministického TS, môže existovať niekoľko rozdielnych výpočtov nedeterministického TS na danom vstupe. Podstatným spôsobom sa mení kritérium, na základe ktorého určujeme, či stroj vstupné slovo akceptoval alebo zamietol. Pretože pripúšťame existenciu viacerých rôznych výpočtov na tom istom vstupe—niektoré z nich môžu byť akceptujúce, zatiaľ čo iné zamietajúce resp. nekonečné—tak je potrebné určiť nové kritérium akceptovania.

Definícia 3.17 (Spôsob akceptovania) *Nech $\mathcal{M}$ je nedeterministický Turingov stroj so vstupnou abecedou Σ. Stroj $\mathcal{M}$* akceptuje *jazyk $L \subseteq \Sigma^*$ práve ak pre každé slovo $w \in L$* existuje *akceptujúci výpočet $\mathcal{M}$ na w.*

Neformálne, nedeterministický stroj akceptuje vstup vtedy, ak *existuje* postupnosť výberov krokov, ktorá vedie k akceptujúcej konfigurácii. Pritom stačí, aby existoval jediný akceptujúci výpočet, všetky ostatné môžu byť zamietajúce. resp. nemusia byť konečné. Naopak, vstup je zamietnutý, ak *žiadna* postupnosť výberu krokov nevedie k akceptujúcej konfigurácii.

Zložitostné miery

Časová zložitosť nedeterministického stroja je funkcia ohraničujúca zhora dĺžku každého výpočtu (tj. nielen akceptujúceho výpočtu).

Definícia 3.18 (Časová zložitosť výpočtu) *Časová zložitosť nedeterministického Turingovho stroja $\mathcal{M}$ na vstupe w je funkcia $t_{\mathcal{M}} : \mathbb{N} \to \mathbb{N} \cup \{\infty\}$ definovaná predpisom*

$$T_{\mathcal{M}}(n) \stackrel{def}{=} \max \{ t_{\mathcal{M}}(w) \mid w \in \Sigma^n \},$$

kde $t_{\mathcal{M}}(w)$ je dĺžka najdlhšieho *výpočtu stroja $\mathcal{M}$ na vstupe w.*

Definícia priestorovej zložitosti nedeterministického TS je zhodná s definíciou priestorovej zložitosti deterministického TS. To znamená, že priestorová zložitosť je funkcia ohraničujúca maximálny počet políčok použitý v *ľubovoľnom* výpočte nedeterministického stroja (vrátane nekonečného). Analogicky ako v prípade deterministických strojov, definujeme priestorovú zložitosť aj pre zariadenia so vstupnou páskou resp. so vstupnou a výstupnou páskou.

Množinu jazykov akceptovaných nedeterministickými Turingovymi strojmi v čase (v priestore) $f(n)$ označujeme $\mathsf{NTIME}(f(n))$ $(\mathsf{NSPACE}(f(n)))$.

Vzťah determinizmu a nedeterminizmu

V tejto časti nás bude zaujímať, aký je vzťah medzi triedou jazykov rozhodovaných deterministickými a nedeterministickými Turingovymi strojmi. Ukážeme, že obe zariadenia sú rovnako silné, tj. že rozhodujú tú istú triedu jazykov.

Veta 3.19 (Vzťah deterministických a nedeterministických TS) *Nech jazyk L je akceptovaný jednopáskovým nedeterministickým Turingovym strojom $\mathcal{N}$ s časovou zložitosťou $f(n)$. Potom L je rozhodovaný deterministickým strojom $\mathcal{D}$ s časovou zložitosťou $2^{\mathcal{O}(f(n))}$.*

Dôkaz Nech $\mathcal{N}$ je nedeterministický TS so vstupnou abecedou Σ, množinou stavov Q a prechodovou reláciou δ. Pre každú dvojicu $(q, a) \in Q \times \Sigma$ uvažujme množinu $C_{q,a} = \left\{ (q', a', D) \;\middle|\; ((q, a), (q', a', D)) \in \delta \right\}$. $C_{q,a}$ je konečná množina, popisujúca všetky možné kroky, ktoré môže jednopáskový stroj $\mathcal{N}$ urobiť z konfigurácie, v ktorej snímaný symbol je a a stav je q. Nech $d = \max_{q,a} |C_{q,a}|$. Číslo d je hornou hranicou pre počet možných vetvení; niekedy sa nazýva aj *stupeň nedeterminizmu* stroja $\mathcal{N}$. Predpokladáme $d > 1$ (prípad $d = 1$ je triviálny; jedná sa o deterministický stroj).

Hlavná myšlienka, na ktorej je založená simulácia stroja $\mathcal{N}$ je takáto: Každý výpočet stroja $\mathcal{N}$ na fixnom slove w možno charakterizovať postupnosťou nedeterministických výberov krokov. Postupnosť t nedeterministických výberov možno prezentovať ako postupnosť t čísel z intervalu $0,1, \dots ,d - 1$. Simulujúci stroj $\mathcal{D}$ uvažuje všetky takéto postupnosti výberov v rastúcom usporiadaní a simuluje stroj $\mathcal{N}$ pre každú z nich. Upozorňujeme, že stroj $\mathcal{D}$ nemôže simulovať len výpočty dĺžky $f(|w|)$, a to preto, že $\mathcal{D}$ nepozná časovú zložitosť stroja $\mathcal{N}$ (t.j. funkciu f).

Predpokladajme, že stroj $\mathcal{D}$ chce simulovať výpočet dĺžky t stroja $\mathcal{N}$, charakterizovaný postupnosťou nedeterministických výberov $(c_1, c_2, \dots , c_t)$. Túto postupnosť má $\mathcal{D}$ zapísanú na svojej druhej páske. Na samotnú simuláciu používa $\mathcal{D}$ tretiu pásku. Keď simuluje i-ty krok výpočtu stroja $\mathcal{N}$, vyberá si z možných krokov práve ten, ktorý je popísaný číslom c_i. Ak simulovaný výpočet stroja $\mathcal{N}$ sa zastaví v akceptujúcej konfigurácii, tak výpočet stroja $\mathcal{D}$ sa tiež zastaví v akceptujúcej konfigurácii. Inak $\mathcal{D}$ pokračuje v simulácii pre nasledujúcu (v rastúcom usporiadaní) postupnosť nedeterministických výberov. Nasledujúcu postupnosť určí jednoducho—stačí číslo zapísané na druhej páske zväčšiť o 1 (v d-árnej sústave).

Ako stroj $\mathcal{D}$ rozpozná, že vstupné slovo má byť zamietnuté, ak nepozná funkciu ohraničujúcu časovú zložitosť stroja $\mathcal{N}$? Ak $\mathcal{D}$ simuluje všetky výpočty zodpovedajúce postupnostiam nedeterministických výberov určitej dĺžky t a každý z nich je už ukončený (t.j. skončil v konfigurácii so stavom q_{reject}alebo q_{stop}), tak $\mathcal{D}$ môže ukončiť simuláciu v zamietajúcej konfigurácii.

Časovú zložitosť navrhnutej simulácie možno zhora ohraničiť číslom $\sum_{t=1}^{f(n)} d^t$ (tj. počet rôznych možných postupností nedeterministických výberov) krát počet krokov potrebných na vygenerovanie jednej postupnosti plus odsimulovanie výpočtu zodpovedajúceho tejto postupnosti. Druhý faktor iste neprevyšuje $\mathcal{O}(f(n))$. Celkove zložitosť simulácie možno vyjadriť ako $2^{\mathcal{O}(f(n))}$. □

Konštrukcia navrhnutá v dôkaze predchádzajúcej vety sa dá rozšíriť aj pre viacpáskové nedeterministické Turingove stroje. Bezprostredným dôsledkom vety 3.19 je inklúzia

$$\mathsf{NTIME}(f(n)) \subseteq \mathsf{DTIME}(2^{f(n)}).$$

Prezentovaná simulácia ukazuje, že výpočtová sila oboch zariadení je rovnaká (inými slovami deterministické a nedeterministické Turingove stroje akceptujú tú istú triedu jazykov). Avšak výpočty na nedeterministických strojoch *môžu* byť podstatne (t.j. až exponenciálne) efektívnejšie v porovnaní s deterministickými výpočtami.

3.4 Churchova téza

V predchádzajúcich kapitolách sme uviedli niekoľko príkladov výpočtových modelov. Ukázali sme, že napriek ich veľkej rozmanitosti je možná ich vzájomná simulácia a teda, že všetky riešia tú istú triedu problémov. Všetky uvedené fakty naznačujú, že Turingove stroje, aj keď majú veľmi jednoduchú dátovú štruktúru a inštrukčnú množinu, sú veľmi silné. Otázkou zostáva, či naozaj dokážu realizovať ľubovoľný algoritmus.

Tvrdíme, že odpoveď na túto otázku je kladná. Po prvý krát tento názor vyjadril A.M. Turing, ktorý poukázal na to, že ním navrhnutá formalizácia skutočne dobre vystihuje intuitívny pojem algoritmu. Jeho tézu by sme mohli vyjadriť takto:

> Každý Turingov stroj, ktorého každý výpočej je konečný, reprezentuje nejaký algoritmus a obrátene, každý algoritmicky riešiteľný problém sa dá riešiť vhodným Turingovym strojom.

Uvedená hypotéza nie je matematickou vetou, pretože hovorí o algoritmoch v nie presne definovanom matematickom zmysle. Preto ani nie je možné ju dokázať; je možné len overovať jej vierohodnosť. Argumenty, ktoré svedčia v jej prospech sú však natoľko presvedčivé, že je bez výhrady prijímaná. Argumenty podporujúce Turingovu tézu vychádzajú z toho, že Turingove stroje nevybočujú z triedy postupov, všeobecne uznávaných za algoritmické. Naopak, každé doteraz navrhnuté výpočtové zariadenie sa dá simulovať Turingovym strojom, t.j téza nebola vyvrátená napriek tomu, že bola formulovaná pred viac ako 60–timi rokmi.

Výpočtový model, ktorý navrhol A.M.Turing (v roku 1936) samozrejme nebol jediným pokusom o formalizáciu pojmu algoritmus. Okrem tých, ktoré sme už prezentovali, spomeňme aspoň práce E.Posta, S.C.Kleeneho, A.Churcha, A.Markova. L.Minského. Skutočnosť, že všetky tieto formalizácie sa ukázali byť navzájom ekvivalentnými, je ďalším faktom, ktorý podporuje uvedenú hypotézu, známu aj pod názvom *Churchova téza*. Jedným z jej hlavných dôsledkov (ak ju prijmeme) je, že ak hovoríme o rozhodnuteľných (t.j. algoritmicky riešiteľných) problémoch, môžeme tak hovoriť bez toho, aby sme upresňovali, vzhľadom k akému modelu sú rozhodnuteľné.

Neskôr, s rozvojom výpočtovej techniky a s tým , ako sa stávali stále dôležitejšími zložitostné charakteristiky algoritmov, vyvstala otázka zložitosti výpočtov na rôznych modeloch. To viedlo k formulovaniu tzv. kvantitatívnej (alebo tiež *polynomiálnej*) *Churchovej tézy* [5]. Podľa nej nielenže všetky „rozumné“ výpočtové modely sú ekvivalentné, ale navyše môžu byť navzájom simulované v polynomiálnom čase[6].

Výpočtové modely spĺňajúce sekvenčnú výpočtovú tézu tvoria tzv. *prvú počítačovú triedu*. Patria sem, okrem iných, všetky uvažované varianty deterministických Turingovych strojov ako aj stroje RAM s logaritmickou cenou inštrukcie. Príkladom modelu, ktorý nepatrí do prvej počítačovej triedy, je stroj RAM s inštrukciou násobenia a s jednotkovou cenou inštrukcie. Hlavným dôsledkom polynomiálnej Churchovej tézy je, že, pokiaľ pracujeme s modelmi z prvej počítačovej triedy, môžeme hovoriť o problémoch riešiteľných v polynomiálnom čase bez toho, aby sme špecifikovali na ktorom modele je výpočet realizovaný.

[5] používa sa tiež názov *sekvenčná výpočtová téza*

[6] Presnejšie, stroj B simuluje stroj A v polynomiálnom čase, ak existuje konštanta k taká, že pre každé celé číslo $t > 1$ a každý vstup, t krokov výpočtu stroja A je možné simulovať nanajvýš t^k krokmi stroja B.

Kapitola 4

Zložitosť a zložitostné triedy

Každá zložitostná trieda je charakterizovaná niekoľkými parametrami. Prvým z nich je zvolený model výpočtu: pre nás sú to Turingove stroje. Druhým parametrom je výpočtový zdroj, na ktorý kladieme ohraničenia: najdôležitejšie (aj keď samozrejme nie jediné) charakteristiky náročnosti výpočtov, na ktoré sa v ďaľšom sústredíme, sú čas a priestor. Ďalším parametrom charakterizujúcim zložitostnú triedu je konkrétny mód: v našom prípade to, či uvažujeme deterministické alebo nedeterministické výpočty (opäť neskôr uvidíme, že to nie sú jediné možnosti). Nakoniec musíme samozrejme špecifikovať hranicu (funkciu f z $\mathbb{N}$ do $\mathbb{N}$) ohraničujúcu zvolený výpočtový zdroj. Zložitostná trieda je množina všetkých jazykov rozhodovaných Turingovymi strojmi v zvolenom móde a takých, že pre každý vstup w zvolená zložitostná charakteristika výpočtu nepresiahne $f(|w|)$.

4.1 Základné charakteristiky

Cieľom tejto časti je zistiť, aká je súvislosť medzi časovou a priestorovou zložitosťou Turingovho stroja a medzi počtom jeho pások, počtom stavov a veľkosťou abecedy. Veta 4.1 hovorí, že, do istej miery, nie je podstatné, koľkopáskový Turingov stroj uvažujeme. Tým, že povolíme väčší počet pások, dosiahneme nanajvýš kvadratické zníženie časovej zložitosti. Veta 4.5 ukazuje, že v prípade priestorovej zložitosti dokonca zníženie počtu pások Turingovho stroja vôbec neovplyvní jeho priestorovú zložitosť. Vety 4.3 a 4.4 zase ukazujú, že zvýšením počtu stavov a symbolov abecedy možno dosiahnuť redukciu zložitosti o konštantný faktor.

4.1.1 Časová zložitosť

Veta 4.1 (Redukcia počtu pások) *K ľubovoľnému k-páskovému Turingovmu stroju $\mathcal{M}$ akceptujúcemu jazyk L s časovou zložitosťou $f(n)$ vieme skonštruovať jednopáskový Turingov stroj $\mathcal{M}'$ s časovou zložitosťou $\mathcal{O}(f(n)^2)$ a akceptujúci ten istý jazyk L.*

Dôkaz Predpokladajme, že $\mathcal{M} = (Q, \Sigma, \Gamma, \triangleright, \sqcup, \delta, q_0)$ je k-páskový deterministický Turingov stroj, $k \geq 2$ (pre $k = 1$ je tvrdenie triviálne). Skonštruujeme s ním ekvivalentný stroj $\mathcal{M}' = (Q', \Sigma', \Gamma', \triangleright', \sqcup, \delta', q_0')$ s jednou páskou, na ktorej budeme simulovať k pások stroja $\mathcal{M}$. Jeden zo spôsobov, ktorým sa to dá zrealizovať, je zapísať na pásku stroja $\mathcal{M}'$ *zreťazenie* obsahu pások stroja $\mathcal{M}$. Okrem toho stroj $\mathcal{M}'$ musí vedieť, aké by boli pozície jednotlivých hláv stroja $\mathcal{M}$. Túto informáciu si stroj môže zapamätať napríklad tak, že symboly, ktoré by snímaly hlavy stroja $\mathcal{M}$, „podčiarkne“. Presnejšie, namiesto symbolu $a \in \Gamma$ zapíše na príslušné políčko symbol $\underline{a}$ (predpokladáme $\underline{a} \notin \Gamma$). Samozrejme, simulácia jedného kroku výpočtu stroja $\mathcal{M}$ si bude vyžadovať niekoľko krokov vpočtu stroja $\mathcal{M}'$.

Prvá fáza simulácie výpočtu stroja $\mathcal{M}$ strojom $\mathcal{M}'$ spočíva v úprave pásky do požadovaného tvaru. Presnejšie, $\mathcal{M}'$ pripojí ku svojmu vstupu reťazec $\triangleleft(\underline{\triangleright}\triangleleft)^{k-1}$. Predpokladáme, že symbol $\triangleleft$ nepatrí do pracovnej abecedy stroja $\mathcal{M}$ a používame ho ako oddeľovač obsahu jednotlivých pások stroja $\mathcal{M}$.

V druhej fáze simuluje $\mathcal{M}'$ výpočet stroja $\mathcal{M}$ krok po kroku. Aby $\mathcal{M}'$ odsimuloval jeden krok, musí dva krát prečítať svoju pásku zľava doprava a späť. Pri prvom prejdení pásky zhromaždí informácie o symboloch, ktoré by čítali jednotlivé hlavy stroja $\mathcal{M}$: sú to práve podčiarknuté symboly. Aby si $\mathcal{M}'$ mohol zapamätať všetky potrebné informácie, musí mať nové stavy, z ktorých každý zodpovedá práve jednej kombinácii stavu stroja $\mathcal{M}$ a k-tici symbolov z abecedy stroja $\mathcal{M}$. Po prvom prejdení svojej pásky teda $\mathcal{M}'$ presne vie, ako musí upraviť obsah svojej pásky, aby zodpovedal novej konfigurácii stroja $\mathcal{M}$. Preto znovu prejde svoju pásku zľava doprava a každý podčiarknutý symbol, prípadne s ním susediaci symbol patrične upraví. Špeciálne treba riešiť situáciu, keď niektorá z hláv stroja $\mathcal{M}$ zapisovala symbol na dovtedy prázdne políčko. Tomu zodpovedá situácia na páske stroja $\mathcal{M}'$, keď je podčiarknutý niektorý zo symbolov $\lhd$. Stroj $\mathcal{M}'$ posunie celý zvyšok reťazca o jedno políčko vpravo a na takto uvoľnené políčko zapíše nový symbol. Simulácia pokračuje, kým sa výpočet stroja $\mathcal{M}$ nezastaví. Stroj $\mathcal{M}'$ upraví svoju pásku tak, aby na nej ostal len reťazec zapísaný na poslednej páske stroja $\mathcal{M}$.

Poďme vyhodnodiť časovú náročnosť práve navrhnutej simulácie. Keďže dĺžka výpočtu stroja $\mathcal{M}$ na vstupe w dĺžky n je nanajvýš $f(n)$, nemôže na žiadnej páske stroja $\mathcal{M}$ byť zapísaný reťazec dlhší než $f(n)$. Takže ani stroj $\mathcal{M}'$ nebude mať na svojej páske nikdy zapísaný reťazec dlhší než $k(f(n)+1)$. Simulácia jedného kroku výpočtu stroja $\mathcal{M}$ si preto vyžiada $4k(f(n)+1)$ (dva prechody pásky tam a späť) plus nanajvýš $k(2k(f(n)+1))$ krokov (vytvorenie prázdneho políčka). Spolu s úvodnou časťou simulácie teda celkový počet krokov stroja $\mathcal{M}'$ neprevýši $\mathcal{O}(k^2 f(\mid w \mid)^2)$. Pretože k' je konštanta nezávislá od dĺžky vstupu, je časová zložitosť stroja $\mathcal{M}'$ práve $\mathcal{O}(f(n)^2)$. □

Uvedená konštrukcia ukazuje, že redukcia počtu pások Turingovho stroja na jednu ovplyvní jeho časovú zložitosť nanajvýš o kvadratický faktor. V prípade, že viacpáskový stroj simulujeme strojom s dvoma páskami, je možné navrhnúť efektívnejšiu simuláciu.

Fakt 4.2 *K ľubovoľnému k-páskovému Turingovmu stroju $\mathcal{M}$ akceptujúcemu jazyk L s časovou zložitosťou $f(n)$ vieme skonštruovať dvojpáskový Turingov stroj $\mathcal{M}'$ akceptujúci ten istý jazyk L s časovou zložitosťou $\mathcal{O}(f(n)\log f(n))$.*

Ďalšou zaujímavou otázkou je, či k ľubovoľnému Turingovmu stroju sa dá skonštruovať stroj, ktorý akceptuje ten istý jazyk, ale pritom je rýchlejší. Nasledujúca veta ukazuje, že vždy sa dá skonštruovať stroj, ktorý je rýchlejší o konštantný faktor.

Veta 4.3 (Veta o urýchlení) *Nech $L \in \mathsf{DTIME}(f(n))$. Potom pre ľubovoľné $\varepsilon > 0$ platí, že $L \in \mathsf{DTIME}(f'(n))$ pre $f'(n) = \varepsilon f(n) + n$.*

Dôkaz Nech $\mathcal{M} = (Q, \Sigma, \Gamma, \rhd, \sqcup, \delta, q_0)$ je k-páskový deterministický Turingov stroj, ktorý rozhoduje jazyk L a ktorého časová zložitosť je $f(n)$. Skonštruujeme $(k+1)$–páskový stroj $\mathcal{M}' = (Q', \Sigma', \Gamma', \rhd', \sqcup, \delta', q_0')$ s časovou zložitosťou $f'(n)$, ktorý simuluje stroj $\mathcal{M}$. Trik, ktorý nám umožní „urýchliť" výpočet stroja $\mathcal{M}$ spočíva v tomto: každý symbol stroja $\mathcal{M}'$ kóduje niekoľko symbolov stroja $\mathcal{M}$. Dôsledkom je, že $\mathcal{M}'$ môže *naraz* simulovať niekoľko krokov výpočtu stroja $\mathcal{M}$.

Popíšeme simuláciu podrobnejšie. Nech m je číslo závisiace len od $\mathcal{M}$ a od ε. Abeceda stroja $\mathcal{M}'$ bude obsahovať, okrem symbolov pracovnej abcedy stroja $\mathcal{M}$ aj všetky m-tice symbolov stroja $\mathcal{M}$, tj. $\Gamma' = \Gamma \cup \Gamma^m$. Na začiatku simulácie posúva $\mathcal{M}'$ prvú hlavu doprava a číta vstup w. Keď prečítal m symbolov zo vstupu, povedzme $a_1 \ldots a_m$, tak na svoju druhú pásku zapíše *jediný* symbol $(a_1, \ldots, a_m) \in \Sigma^m$. V prípade, že $\mathcal{M}'$ dočítal vstupný reťazec a ostala mu nedokončená m-tica, doplní ju sprava na m symbolov pridaním potrebného počtu symbolov $\sqcup$. Po prečíataní celého vstupu je teda na druhej páske zapísaný „zhustený" vstup, jeho dĺžka je $\lceil \frac{|w|}{m} \rceil$. Obsahy ostatných pások sú nezmenené. Od tohoto okamihu bude stroj $\mathcal{M}'$ narábať s druhou páskou ako s páskou obsahujúcou vstup; prvú pásku už nebude používať.

$\mathcal{M}'$ je pripravený začať vlastnú simuláciu výpočtu stroja $\mathcal{M}$. Vždy simuluje m krokov stroja $\mathcal{M}$ v jednom cykle, pozostávajúcom zo šiestich krokov. Na začiatku každého cyklu je stav stroja $\mathcal{M}'$ $(k+1)$-tica $(q, j_1, \ldots, j_k)$. Pritom q je stav stroja $\mathcal{M}$ v prvom z m simulovaných krokov a $j_i \leq m$ je presná pozícia hlavy stroja $\mathcal{M}$ na i-tej páske *v rámci* snímanej m-tice. Presnejšie: ak v niektorom okamihu simulácie bude i-ta hlava stroja $\mathcal{M}$ snímať l-te políčko vpravo od $\triangleright$, tak $i+1$-vá hlava stroja $\mathcal{M}'$ bude snímať $\lceil\frac{l}{m}\rceil$-te políčko vpravo od $\triangleright$ a hodnota j_i bude (l mod m).

Na začiatku cyklu stroj $\mathcal{M}'$ posunie všetky svoje hlavy o jedno políčko vľavo, potom dva krát o jedno políčko vpravo a nakoniec o jedno políčko vľavo. Teraz si $\mathcal{M}'$ v stave pamätá všetky symboly, čítané jednotlivými hlavami stroja $\mathcal{M}$ spolu so symbolmi, ktoré sú zapísané vedľa nich. Dôležité je si uvedomiť, že na základe tejto informácie vie $\mathcal{M}'$ presne predvídať najbližších m krokov stroja $\mathcal{M}$. Dôvodom je, že v priebehu m krokov sa žiadna hlava stroja $\mathcal{M}$ neposunie o viac ako m políčok vpravo alebo vľavo. Obsah týchto políčok je zapísaný na políčkach, ktoré práve čítajú hlavy stroja $\mathcal{M}'$ a na políčkach s nimi bezprostredne susediacimi. Preto $\mathcal{M}'$ vie, ako má zmeniť obsahy svojich pások tak, aby zodpovedali konfigurácii, ktorú dosiahne $\mathcal{M}$ po m simulovaných krokoch. Stroju $\mathcal{M}'$ na to stačia dva kroky: stačí zmeniť obsah snímaného políčka a nanajvýš jedného zo susedných políčok (nikdy nie oboch naraz). Celkove na simuláciu m krokov stroja $\mathcal{M}$ stačilo stroju $\mathcal{M}'$ nanajvýš 6 krokov.

Simulácia skončí, ak sa zastaví výpočet stroja $\mathcal{M}$. Analyzujme časovú zložitosť výpočtu stroja $\mathcal{M}'$ na vstupe w. Prípravná fáza zahŕňa $|w|+1+\lceil\frac{|w|}{m}\rceil$ krokov (hlavu na druhej páske musíme posunúť na doľava až na ľavú koncovú značku). Počet simulačných cyklov je $\lceil\frac{f(|w|)}{m}\rceil$. Časová zložitosť stroja $\mathcal{M}'$ je preto $f'(n) = n + 1 + \lceil\frac{n}{m}\rceil + 6\lceil\frac{f(n)}{m}\rceil$. Preto stačí zvoliť konštantu m tak, aby $f'(n) \leq \varepsilon f(n) + n$. □

Upozorňujeme, že „urýchlenie“ výpočtov bolo dosiahnuté na úkor nárastu počtu stavov a veľkosti abecedy. Ak by sme fixovali abecedu Turingovho stroja, tak podobný efekt by nebol možný. Takže skutočnosť, že pri uvažovaní časovej zložitosti nie je rozdiel medzi ohraničením $f(n)$ a $c \cdot f(n)$ je skôr vlastnoť uvažovaného výpočtového modelu, než vlastnosť algoritmov.

Veta o urýchlení sa dá, úplne analogickým postupom, dokázať aj pre nederministické TS. Ako dôsledok dostávame, že pre ľubovoľné $c > 0$ platia rovnosti

$$\mathsf{DTIME}(f(n)) = \mathsf{DTIME}(cf(n) + n)$$

$$\mathsf{NTIME}(f(n)) = \mathsf{NTIME}(cf(n) + n).$$

Význam práve dokázanej vety je v tom, že nám dovoľuje používať pri špecifikácii časových zložitostných tried $\mathcal{O}$ notáciu. Napríiklad, ak časová zložitosť TS je plynóm stupňa k, tak veta 4.3 zaručuje existenciu ekvivalentného stroja s časovou zložitosťou n^k. Teda ak nejaký jazyk je rozhodovaný v polynomiálnom čase, tak patrí do triedy $\mathsf{DTIME}(n^k)$ pre nejaké prirodzené číslo k. Zjednotením všetkých tried $\mathsf{DTIME}(n^k)$ pre $k > 0$ dostaneme triedu všetkých jazykov rozhodovaných v polynomiálnom čase. Podobne pre ostatné typy funkcií.Pre najčastejšie uvažované zložitostné triedy definujeme označenie:

$$\begin{array}{lcl}
\mathsf{P} & = & \bigcup_{k>0} \mathsf{DTIME}(n^k) \\
\mathsf{NP} & = & \bigcup_{k>0} \mathsf{NTIME}(n^k) \\
\mathsf{ETIME} & = & \bigcup_{k>0} \mathsf{DTIME}(2^{kn}) \\
\mathsf{NETIME} & = & \bigcup_{k>0} \mathsf{NTIME}(2^{kn}) \\
\mathsf{EXPTIME} & = & \bigcup_{k>0} \mathsf{DTIME}(2^{n^k}) \\
\mathsf{NEXPTIME} & = & \bigcup_{k>0} \mathsf{NTIME}(2^{n^k})
\end{array}$$

4.1.2 Priestorová zložitosť

Takisto ako v prípade časovej zložitosti, aj pre priestor si môžeme klásť otázku, či sa k ľubovoľnému stroju dá skonštruovať ekvivalentný stroj s menšou priestorovou zložitosťou. Odpoveď na túto otázku dáva Veta 4.4.

Veta 4.4 (Veta o kompresii priestoru) *Nech $L \in \mathsf{DSPACE}(f(n))$. Potom pre ľubovoľné $\varepsilon > 0$, $L \in \mathsf{DSPACE}(\varepsilon f(n))$.*

Dôkaz Nech $\mathcal{M}$ je TS so vstupnou páskou a s priestorovou zložitosťou $f(n)$. Konštrukcia $f'(n)$–priestorovo ohraničeného stroja $\mathcal{M}'$, simulujúceho stroj $\mathcal{M}$, je založená na podobnom triku, ako sme použili pri konštrukcii „rýchlejšieho“ stroja: každý symbol stroja $\mathcal{M}'$ kóduje niekoľko symbolov stroja $\mathcal{M}$. Technická realizácia je zhodná s postupom použitým v dôkaze vety 4.3. □

Veta o kompresii priestoru sa dá rozšíriť aj pre nederministické TS.

Ako dôsledok viet 4.1 a 4.4 dostávame nasledovné tvrdenie, ktoré ukazuje, že zmena počtu pások Turingovho stroja neovplyvní jeho priestorovú zložitosť.

Veta 4.5 *K ľubovoľnému k–páskovému Turingovmu stroju $\mathcal{M}$ so vstupnou a výstupnou páskou akceptujúcemu jazyk L s priestorovou zložitosťou $f(n)$, vieme skonštruovať Turingov stroj $\mathcal{M}'$ s jednou pracovnou páskou, akceptujúci ten istý jazyk L a s tou istou priestorovou zložitosťou.*

Posledné dve uvedené vety opodstatňujú zavedenie $\mathcal{O}$ notácie aj pri špecifikácii priestorových zložitostných tried. Používame analogické označenie ako pre časové triedy:

$$\begin{array}{lcl} \mathsf{DLOG} & = & \mathsf{DSPACE}(\log n) \\ \mathsf{NLOG} & = & \mathsf{NSPACE}(\log n) \\ \mathsf{PSPACE} & = & \bigcup_{k>0} \mathsf{DSPACE}(n^k) \\ \mathsf{NPSPACE} & = & \bigcup_{k>0} \mathsf{NSPACE}(n^k) \end{array}$$

4.2 Hierarchie zložitostných tried

V predchádzajúcej časti sme špecifikovali pojem zložitostnej triedy. Ďalším predmetom nášho skúmania budú samotné zložitostné triedy a vzťahy medzi nimi. Sústredíme sa na nasledovné dve otázky. Prvá z nich sa týka možnosti ohraničiť zhora časovú resp. priestorovú zložitosť všetkých rekurzívnych jazykov. Inými slovami, pýtame sa, či existuje taká funkcia f, že každý rekurzívny jazyk patrí do triedy $\mathsf{DTIME}(f(n))$ resp. $\mathsf{DSPACE}(f(n))$. Ukážeme, že odpoveď na túto otázku je negatívna. S ňou prirodzene súvisí druhá otázka a to, o koľko musíme zväčšiť výpočtový zdroj uvažovaného zariadenia, aby zariadenie rozpoznávalo väčšiu triedu jazykov? Vety o urýchlení a o kompresii priestoru (vety 4.3 a 4.4) ukazujú, že nárast zložitosti o konštantný faktor nezmení zodpovedajúcu zložitostnú triedu. Výsledky prezentované v kapitolách 4.2.1 a 4.2.2 presne ukazujú, o koľko musíme zvýšiť priestorovú resp. časovú zložitosť Turingovych strojov, aby dokázali počítať komplikovanejšie úlohy.

Skôr než budeme prezentovať výsledky o hierarchiách, musíme vyriešiť niektoré (technické) problémy týkajúce sa zložitosti TS a zložitostných funkcií.

Aké funkcie môžeme uvažovať ako zložitostné miery? V princípe každá funkcia definovaná na množine prirodzených čísel môže byť použitá na definovanie zložitostnej triedy. Takýto všeobecný prístup dovoľuje, aby sme ako zložitostné hranice uvažovali aj veľmi komplikované funkcie, ktoré samotné nemôžu byť vypočítané v hraniciach, ktoré špecifikujú. Použitie takýchto funkcií vedie k neočakávaným a zavádzajúcim výsledkom. Preto definujeme pojem *konštruovateľnej funkcie*. V ďaľšom texte budeme pod zložitostnou funkciou vždy rozumieť konštruovateľnú funkciu—s výnimkou vety 4.16, kde budeme ilustrovať vlastnosti zložitostných tried špecifikovaných funkciami, ktoré nie sú konštruovateľné.

Definícia 4.6 (Konštruovateľná funkcia) *Nech f je funkcia, $f : \mathbb{N} \to \mathbb{N}$. Funkciu f nazveme* konštruovateľnou *práve ak existuje k-páskový deterministický Turingov stroj $\mathcal{M}_f$ so vstupnou a výstupnou páskou taký, že pre každý vstup x dĺžky n, je $\mathcal{M}_f(x) = \$^{f(n)}$, časová zložitosť stroja $\mathcal{M}_f$ je $\mathcal{O}(n + f(n))$ a priestorová zložitosť stroja $\mathcal{M}_f$ je $\mathcal{O}(f(n))$.*

Príklad 4.7 *Nasledujúce funkcie sú konštruovateľné: $f(n) = c$ (c je fixovaná konštanta), $f(n) = n$, $f(n) = \lceil \log n \rceil$. Ľahko nahliadneme, že ak f a g sú konštruovateľné funkcie, tak aj $f+g$, $f \cdot g$ a 2^g sú konštruovateľné (Turingov stroj najprv vypočíta funkcie f a g a potom výslednú funkciu). Z toho napríklad vyplýva, že aj funkcie $\log n^2$, $n \log n$, $n^5 + (\log n)^3$, $n!$, $\sqrt{n}$ sú konštruovateľné. Podotýkame, že pod každou z uvedených funkcií $f(n)$ rozumieme v skutočnosti funkciu $\lceil f(n) \rceil$ (pozri kap. 1).*

Konštruovateľné funkcie umožňujú konštrukciu takých strojov, pre ktoré časová zložitosť všetkých výpočtov na vstupoch rovnakej dĺžky je totožná. Tieto tzv. presné TS nie sú dôležité samy o sebe, sú skôr technickým prostriedkom, ktorý využijeme neskôr pri rozširovaní TS napr. o paralelizmu a náhodnosť.

Formálne, (deterministický alebo nedeterministcký) Turingov stroj $\mathcal{M}$ nazývame *presný*, ak existuje funkcia f taká, že každý výpočet stroja $\mathcal{M}$ na vstupe w sa zastaví *presne* po $f(|w|)$ krokoch.

Lema 4.8 (Existencia presného TS) *Nech (deterministický alebo nedeterministický) Turingov stroj $\mathcal{M}$ rozhoduje jazyk L a jeho časová zložitosť je $f(n)$, kde $f(n)$ je konštruovateľná funkcia. Potom existuje* presný *Turingov stroj $\mathcal{M}'$ rozhodujúci ten istý jazyk v čase $\mathcal{O}(f(n))$.*

Dôkaz V obidvoch prípadoch (determinizmus, nedeterminizmus) začína $\mathcal{M}'$ svoj výpočet na vstupe w výpočtom hodnoty $f(|w|)$ a to tak, že simuluje stroj $\mathcal{M}_f$ počítajúci f. V ďalšom výpočte $\mathcal{M}'$ používa vypočítanú hodnotu $f(|w|)$ ako *počítadlo*. $\mathcal{M}'$ simuluje výpočet $\mathcal{M}$, pričom za každý krok stroja $\mathcal{M}$ posunie hlavu na počítadle o jedno políčko doprava. $\mathcal{M}'$ ukončí simuláciu výpočtu stroja $\mathcal{M}$ práve vtedy, keď hlava čítajúca počítadlo prvý krát narazí na prázdny symbol. Ak simulovaný výpočet skončil po menej ako $f(|w|)$ krokoch, tak $\mathcal{M}'$ pokračuje ďalej vo výpočte, pričom nemení obsah svojich pások s výnimkou počítadla, ktorého hodnotu znižuje. Ak $\mathcal{M}$ je nedeterministický, tak všetky možné výpočty stroja $\mathcal{M}'$ budú rovnako časovo náročné, pretože táto náročnosť závisí len od hodnoty, ktorú $\mathcal{M}'$ *deterministicky* vypočíta v prvej fáze simulácie. Časová zložitosť stroja $\mathcal{M}'$ je zložitosť výpočtu $f(n)$ plus zložitosť simulácie a teda nepresiahne $\mathcal{O}(f(n))$. □

Zavedenie pojmu konštruovateľná funkcia má ešte ďalší zaujímavý dôsledok. Umožní nám totiž predpokladať, že každý výpočet priestorovo ohraničeného Turingovho stroja je konečný. Pripomíname, že podľa definície priestorovej zložitosti môže aj nekonečný výpočet mať konečnú priestorovú zložitosť.

Lema 4.9 *Nech $\mathcal{M}$ je (ne)deterministický Turingov stroj s priestorovou zložitosťou f, kde f je konštruovateľná a $f(n) \geq \log n$. Potom existuje $f(n)$–priestorovo ohraničený (ne)deterministický Turingov stroj $\mathcal{M}'$ rozhodujúci ten istý jazyk a ktorého každý výpočet na každom vstupe je konečný.*

Dôkaz Nech $\mathcal{M} = (Q, \Sigma, \Gamma \triangleright, \sqcup, \delta, q_0)$ je k–páskový Turingov stroj so vstupnou páskou a s priestorovou zložitosťou $f(n)$. V každom výpočte na vstupe dĺžky n je počet neprázdnych políčiek na každej pracovnej páske stroja $\mathcal{M}$ nanajvýš $f(n)$. Preto aj počet rôznych konfigurácii, ktorými môže stroj prechádzať vo výpočtoch na vstupoch dĺžky n, je konečný a ohraničený hodnotou $p(n) = |Q| \cdot (n+2) \cdot \cdot f(n)^{k-1} \cdot (|\Gamma|^{f(n)})^{k-1}$ (v poradí: počet rôznych stavov, možné pozície hlavy na vstupnej páske a na pracovných páskach, počet rôznych obsahov pracovných pások). Navrhneme stroj $\mathcal{M}'$ s požadovanými vlastnosťami. Na vstupe w dĺžky n stroj $\mathcal{M}'$ najprv simuluje výpočet stroja $\mathcal{M}_f$, ktorý počíta funkciu f. Ak $\mathcal{M}'$ pozná hodnotu $f(|w|)$ a konštanty $|Q|$ a $|\Gamma|$, tak vie vypočítať hodnotu $p(n)+1$. Číslo $p(n)+1$ si zapíše $\mathcal{M}'$ *binárne* na jednu zo svojich pások; v ďalšom ju bude používať ako počítadlo. Dĺžka binárneho zápisu nepresiahne $\lceil \log(p(n)+1) \rceil$. $\mathcal{M}'$ pokračuje vo výpočte deterministickou (nedeterministickou) simuláciou výpočtu stroja $\mathcal{M}$ na vstupe w, pričom za každý krok zníži hodnotu počítadla o 1. Ak sa výpočet stroja $\mathcal{M}$ zastaví, tak aj $\mathcal{M}'$ ukončí svoj výpočet a $\mathcal{M}(w) = \mathcal{M}'(w)$. Ak $\mathcal{M}'$ znížil obsah počítadla na 0, tak ukončí svoj výpočet a vstup neakceptuje. Vyprázdnenie počítadla totižto indikuje, že v simulovanom výpočte stroja $\mathcal{M}$ sa objavili dve rovnaké konfigurácie. Ak $\mathcal{M}$ je deterministický, tak simulovaný výpočet je nekonečný (úsek výpočtu medzi rovnakými konfiguráciami sa bude stále opakovať) a preto $\mathcal{M}'$ môže vstup zamietnuť. V prípade, že $\mathcal{M}$ je nedeterministický, môže simulovaný výpočet stroja $\mathcal{M}$ skončiť akceptujúcou konfiguráciou. V takomto prípade však určite existuje kratší (dokonca kratší než $p(n)+1$) výpočet stroja $\mathcal{M}$, končiaci tou istou akceptujúcou konfiguráciou (stačí vynechať úsek výpočtu medzi dvomi zhodnými konfiguráciami). Preto $\mathcal{M}'$ aj v tomto prípade môže vstup zamietnuť. □

Fakt 4.10 *Veta 4.9 platí aj v prípade, že funkcia f nie je konštruovateľná.*

Teraz sme už pripravení podať odpoveď na prvú z otázok formulovaných v úvode tejto kapitoly.

Veta 4.11 *Ku každej konštruovateľnej funkcii $f(n)$ existuje rekurzívny jazyk L taký, že $L \notin \mathsf{DTIME}(f(n))$. Analogicky, ku každej konštruovateľnej funkcii $f(n)$ existuje rekurzívny jazyk L taký, že $L \notin \mathsf{DSPACE}(f(n))$.*

Dôkaz Ukážeme platnosť tvrdenia pre čas; argumenty pre priestor sú podobné.

Nech $f(n)$ je konštruovateľná funkcia a $\mathcal{F}$ deterministický Turingov stroj, ktorý ju počíta. Skonštruujeme rekurzívny jazyk L_f nepatriaci do $\mathsf{DTIME}(f(n))$.

V kapitole 1 sme zaviedli kódovanie Turingovych strojov reťazcami nad abecedou $\{0,1\}$. Každý binárny reťazec môžeme považovať za kód nejakého Turingovho stroja; v prípade, že reťazec nemá predpokladaný tvar, považujeme ho za kód stroja rozpoznávajúceho prázdnu množinu. Označme $\mathcal{M}_x$ viacpáskový deterministický Turingov stroj, ktorého kódom je práve binárny reťazec x. Ďalej definujme jazyk L_f nad abecedou $\{0,1\}$ nasledovne:

$$L_f \stackrel{\text{def}}{=} \{x \mid \text{ výpočet } \mathcal{M}_x \text{ na } x \text{ sa zastaví po nanajvýš } f(|x|) \text{ krokoch v zamietajúcej konfigurácii}\}.$$

Ukážeme, že L_f je rekurzívny jazyk. Stačí, ak skonštruujeme deterministický Turingov stroj $\mathcal{T}$ rozhodujúci L_f. Stroj $\mathcal{T}$ pracuje takto: pre vstup w dĺžky n simuluje najprv stroj $\mathcal{F}$ ($\mathcal{F}$ počíta funkciu f). Potom $\mathcal{T}$ na jednu zo svojich pracovných pások prepíše w a ďalej ho chápe ako kód stroja, ktorý má simulovať. Nakoniec $\mathcal{T}$ simuluje prvých $f(|w|)$ krokov výpočtu $\mathcal{M}_w$ na w. $\mathcal{T}$ akceptuje, ak simulovaný výpočet $\mathcal{M}_w$ sa zastavil v zamietajúcej konfigurácii.

Nakoniec overíme, že $L_f \notin \mathsf{DTIME}(f(n))$. Predpokladajme opak, teda že existuje deterministický TS $\mathcal{M}$ rozhodujúci L_f v čase $f(n)$. Stroj $\mathcal{M}$ má kód, označme ho x, t.j $\mathcal{M} = \mathcal{M}_x$. Ako vyzerá výpočet stroja $\mathcal{M}$ na vstupe x? Podľa predpokladu má dĺžku nepresahujúcu $f(|x|)$ a končí buď v akceptujúcej alebo zamietajúcej konfigurácii. Preverme obe možnosti.

Nech $\mathcal{M}$ akceptuje x. Z definície jazyka L_f máme, že $x \notin L_f$ a to je spor s predpokladom, že $\mathcal{M}$ rozhoduje L_f.

Nech $\mathcal{M}$ zamieta x. Z definície jazyka L_f máme, že $x \in L_f$ a to je spor s predpokladom, že $\mathcal{M}$ rozhoduje L_f.

Obidve možnosti viedli k sporu a preto predpoklad, že L_f je rozhodovaný $f(n)$ časovo ohraničeným strojom $\mathcal{M}$, musel byť nesprávny. □

Metóda použitá v predchádzajúcom dôkaze sa nazýva metóda diagonalizácie a je (ako uvidíme ďalej) v teórii zložitosti veľmi často používanou technikou.

Ak $f(n)$ je konštruovateľná funkcia, tak veta 4.11 implikuje existenciu rekurzívneho jazyka L, ktorý nepatrí do $\mathsf{DTIME}(f(n))$. Nech $\hat{f}(n)$ je časová zložitosť stroja, ktorý rozhoduje jazyk L (určite existuje, lebo L je rekurzívny). Položme $f'(n) = \max\{f(n), \hat{f}(n)\}$. Potom určite $\mathsf{DTIME}(f(n)) \subset \mathsf{DTIME}(f'(n))$ pretože L patrí do druhej ale nepatrí do prvej z uvedených tried. Znamená to, že hierarchia časových zložitostných tried je nekonečná. Podobný výsledok sa dá (analogickým postupom) dokázať aj pre deterministické priestorové zložitostné triedy a pre nedeterministické časové a priestorové zložitostné triedy.

4.2.1 Priestorová hierarchia

Vieme už (veta 4.11), že ku každej konštruovateľnej funkcii $f(n)$ existuje funkcia $f'(n)$, že $\mathsf{DSPACE}(f(n)) \subset \mathsf{DSPACE}(f'(n))$. Otázkou je, ako minimálne sa musia líšiť funkcie $f(n)$ a $f'(n)$. Ukážeme teraz, že pre konštruovateľné funkcie stačí, aby sa obe funkcie líšili veľmi málo.

Veta 4.5 nám dovoľuje obmedziť sa v nasledujúcich úvahách na stroje so vstupnou, výstupnou a jednou pracovnou páskou.

Veta 4.12 (Priestorová hierarchia) *Nech $f_1, f_2 : \mathbb{N} \to \mathbb{N}$ sú funkcie s nasledovnými vlastnosťami:*

1. $f_1(n) \geq \log n$ *pre všetky* $n \in \mathbb{N}$,
2. f_2 *je konštruovateľná*
3. $f_1(n) = o(f_2(n))$

Potom $\mathsf{DSPACE}(f_1(n)) \subset \mathsf{DSPACE}(f_2(n))$.

Dôkaz Na dôkaz tohoto tvrdenia použijeme (podobne ako vo vete 4.11) metódu diagonalizácie. Cieľom je skonštruovať jazyk L_d, ktorý nie je rozhodovaný žiadnym TS uvažovaného typu (v našom prípade sú to $f_1(n)$-priestorovo ohraničené deterministické TS). Najjednoduchšia verzia tejto techniky vychádza z efektívneho usporiadania všetkých Turingovych strojov uvažovaného typu do postupnosti $\mathcal{T}_0, \mathcal{T}_1, \mathcal{T}_2, \ldots$. Pre každé $i \in \mathbb{N}$ potom vyberieme slovo $x_i \in \{0,1\}^*$ tak, aby pre $i \neq j$ bolo $x_i \neq x_j$. Slovo x_i je *potenciálnym kandidátom*, ktorý rozlíši jazyk L_d (hľadaný jazyk) a $L(\mathcal{T}_i)$ (jazyk rozhodovaný strojom $\mathcal{T}_i$). Definujeme diagonálny jazyk L_d takto:

$$L_d \stackrel{\text{def}}{=} \{\, x_i \mid x_i \notin L(\mathcal{T}_i),\ i \in \mathbb{N} \,\}\,.$$

Pre takto definovaný jazyk L_d skutočne platí, že x_i rozlišuje jazyky L_d a $L(\mathcal{T}_i)$—nazývame ho preto *úspešným kandidátom*.

Ak chceme techniku diagonalizácie použiť pre dôkaz nášho tvrdenia, je potrebné ju zovšeobecniť, a to hneď z dvoch dôvodov. Za prvé, nevieme efektívne vymenovať postupnosť všetkých $f_1(n)$ priestorovo ohraničených Turingovych strojov, pretože problém určiť, či daný Turingov stroj je $f_1(n)$ priestorovo ohraničený, je nerozhodnuteľný. Preto musíme vziať postupnosť všetkých Turingovych strojov a nájsť kandidáta pre každý z nich. Samozrejme, ak stroj $\mathcal{T}$ nie je $f_1(n)$ priestorovo ohraničený, tak kandidát môže byť neúspešný, pretože nepotrebujeme, aby sa preňho diagonálny jazyk L_d a jazyk $L(\mathcal{T})$ naozaj líšili. Stačí nám zabezpečiť, aby kandidáti rozlišovali všetky $f_1(n)$ priestorovo ohraničené stroje od diagonálneho jazyka.

Druhý problém súvisí so zložitosťou simulácie. Jazyk L_d má patriť do triedy $\mathsf{DSPACE}(f_2(n))$, čo znamená, že musí existovať $f_2(n)$–priestorovo ohraničený Turingov stroj $\mathcal{M}$ rozhodujúci L_d. Stroj $\mathcal{M}$ má konečnú abecedu ale pritom musí byť schopný simulovať prácu ľubovoľného $f_1(n)$–priestorovo ohraničeného stroja. Preto je potrebné abecedu Γ simulovaného stroja kódovať binárne—na zakódovanie jedného symbolu potrebujeme práve $\lceil \log |\Gamma| \rceil$ bitov. Na simuláciu $f_1(n)$–priestorovo ohraničeného stroja potrebuje preto $\mathcal{M}$ priestor $\lceil \log |\Gamma| \rceil \cdot f_1(n)$. Z predokladu o vzťahu medzi funkciami f_1 a f_2 vieme, že pre konečne veľa hodnôt n môže byť $\lceil \log |\Gamma| \rceil \cdot f_1(n) > f_2(n)$. Preto pre slová malej dĺžky nemusí byť $\mathcal{M}$ schopný simulovať stroj s veľkou pracovnou abecedou. Nemôžeme fixovať kandidáta x_i pre $\mathcal{T}_i$ bez toho, aby sme mali istotu, že simulácia bude môcť prebehnúť úspešne, t.j. že $\lceil \log \mid \Gamma_i \mid \rceil \cdot f_1(|x_i|) \leq f_2(|x_i|)$, kde Γ_i je abeceda stroja $\mathcal{T}_i$.

Problém sa dá vyriešiť tak, že pre každý stroj $\mathcal{T}_i$ vyberieme nekonečnú množinu potenciálnych kandidátov. Potom môžemem mať istotu, že existuje kandidát w taký, že $\lceil \log |\Sigma_i| \rceil \cdot f_1(|w|) \leq f_2(|w|)$ a stroj $\mathcal{M}$ môže simulovať stroj $\mathcal{T}_i$ na vstupe w a akceptovať (zamietnuť) práve ak $\mathcal{T}_i$ zamietol (akceptoval), tj. w je úspešný kandidát.

Teraz sme pripravení podať formálny dôkaz vety. Skonštruujeme $f_2(n)$-priestorovo ohraničený dvojpáskový deterministický Turingov stroj $\mathcal{M}$, ktorý rozhoduje jazyk $L \subseteq \{0,1\}^*$ nepatriaci do $\mathsf{DSPACE}(f_1(n))$.

Pre vstup w dĺžky n stroj $\mathcal{M}$ najprv simuluje stroj $\mathcal{F}_2$, ktorý počíta funkciu f_2 a označí na svojej pracovnej páske práve $f_2(n)$ políčok. V ďalšom výpočte $\mathcal{M}$ nikdy nepoužije neoznačené políčko; pri pokuse o zápis na neoznačené políčko sa výpočet zastaví v zamietajúcej konfigurácii. Tým máme zaručené, že $\mathcal{M}$ je $f_2(n)$ priestorovo ohraničený.

Každý vstupný reťazec w stroja $\mathcal{M}$ sa dá napísať v tvare $w = 1^*0w'$. V nasledujúcej časti výpočtu $\mathcal{M}$ simuluje výpočet stroja $\mathcal{M}_{w'}$ na vstupe w. $\mathcal{M}_{w'}$ je stroj, ktorého kód je práve w'. Za predpokladu, že stroj $\mathcal{M}_{w'}$ je $f_1(n)$-priestorovo ohraničený, stačí stroju $\mathcal{M}$ na jeho úspešnú simuláciu priestor $\lceil \log t \rceil \cdot f_1(n)$, kde t je počet symbolov pracovnej abecedy stroja $\mathcal{M}_{w'}$. Stroj $\mathcal{M}$ akceptuje vstup w práve vtedy, ak úspešne dokončil simuláciu výpočtu stroja $\mathcal{M}_{w'}$ a $\mathcal{M}_{w'}$ neakceptoval vstup w.

Zrejme jazyk L rozhodovaný strojom $\mathcal{M}$ patrí do $\mathsf{DSPACE}(f_2(n))$. Ukážeme, že L nepatrí do $\mathsf{DSPACE}(f_1(n))$.

Predpokladajme opak, t.j. že existuje $f_1(n)$–priestorovo ohraničený stroj $\overline{\mathcal{M}}$ akceptujúci jazyk L. Nech t je počet prvkov pracovnej abecedy stroja $\overline{\mathcal{M}}$. Na základe Faktu 4.10 môžeme predpokladať, že $\overline{\mathcal{M}}$ rozhoduje jazyk L (t.j. každý jeho výpočet je konečný). Nech $\overline{w}$ je kód stroja $\overline{\mathcal{M}}$. Pretože $f_1(n) = o(f_2(n))$, existuje číslo n_t také, že pre všetky $n \geq n_t$ je

$$\lceil \log t \rceil \cdot f_1(n) \leq f_2(n).$$

Fixujme $n > \max\{n_t, |\overline{w}| + 1\}$. Uvažujme slovo $w = 1^{n-|\overline{w}|-1}0\overline{w}$. Pre vstup w má $\mathcal{M}$ dostatok priestoru na simuláciu výpočtu stroja $\overline{\mathcal{M}}$ na vstupe w. Stroj $\mathcal{M}$ akceptuje w práve vtedy, keď stroj $\overline{\mathcal{M}}$ zamietne w. Naopak, ak $\mathcal{M}$ zamieta w, tak $\overline{\mathcal{M}}$ ho akceptuje. To je spor s predpokladom $L \in \mathsf{DSPACE}(f_1(n))$. □

Spôsob, akým je definovaný jazyk L_d, rozlišujúci triedy $\mathsf{DTIME}(f_1(n))$ a $\mathsf{DTIME}(f_2(n))$, je odlišný od spôsobu, akým sa bežne jazyky definujú. Obvykle je jazyk definovaný ako množina reťazcov majúcich nejakú špecifikovanú vlastnosť. Jazyk L_d je naopak popísaný prostredníctvom TS, ktorý ho rozhoduje, tj. je popísaný algoritmicky. V ďalších kapitolách nájdeme aj explicitne (nealgoritmicky) definované jazyky rozlišujúce rôzne zložitostné triedy.

Veta 4.12 platí aj v prípade, že nepožadujeme $f_2(n) \geq \log n$. Podobný výsledok pre nedeterministické priestorové zložitostné triedy sa nedá dokázať tak priamočiaro. Na to, aby stroj $\mathcal{M}$ mohol rozhodnúť o akceptovaní alebo neakceptovaní strojom $\mathcal{M}_w$, musel by preveriť *všetky* jeho výpočty na danom vstupe, na čo mu nemusí stačiť vymedzený priestor. Bližšie sa k tejto otázke vrátime v časti 4.4.

4.2.2 Časová hierarchia

Deterministická časová hierarchia nie je tak tesná, ako priestorová. Je to dané tým, že simulácia viacpáskového Turingovho stroja dvojpáskovým strojom prináša so sebou spomalenie výpočtu o logaritmický faktor (pozri fakt 4.5).

Veta 4.13 (Časová hierarchia) *Nech $f_1, f_2 : \mathbb{N} \to \mathbb{N}$ sú funkcie s nasledovnými vlastnosťami:*

1. *f_2 je konštruovateľná*
2. *$f_1(n)\log(f_1(n)) = o(f_2(n))$*

Potom existuje jazyk, ktorý patrí do $\mathsf{DTIME}(f_2(n))$, ale nepatrí do $\mathsf{DTIME}(f_1(n))$.

Dôkaz Dôkaz je vedený podobným spôsobom ako pre priestorovú hierarchiu. Navrhneme $f_2(n)$–časovo ohraničený stroj $\mathcal{M}$ rozpoznávajúci jazyk $L \in \{0,1\}^*$. Stroj $\mathcal{M}$ na vstupe w najprv vypočíta hodnotu $f_2(|w|)$. Vstup w môžeme zapísať v tvare $1^*0w'$. V ďalšej časti výpočtu bude $\mathcal{M}$ chápať w' ako kód Turingovho stroja, nazvime ho $\mathcal{M}_{w'}$. $\mathcal{M}$ simuluje prvých nanajvýš $f_2(n)$ krokov výpočtu $\mathcal{M}_{w'}$ na vstupe w. Stroj $\mathcal{M}$ akceptuje vstup w práve vtedy, ak úspešne dokončil simuláciu výpočtu stroja $\mathcal{M}_{w'}$ a $\mathcal{M}_{w'}$ neakceptoval vstup w.

Predpokladajme, že simulovaný stroj $\mathcal{M}_w$ je $f_1(n)$–časovo ohraničený. Stroj $\mathcal{M}_w$ môže mať ľubovoľný počet pások, zatiaľ čo $\mathcal{M}$ má fixovaný, vopred určený počet pások. Preto $\mathcal{M}$ musí simulovať $\mathcal{M}_w$ s použitím menšieho počtu pások, než mal k dispozícii $\mathcal{M}_w$. Podľa Faktu 4.5 vieme, že takáto simulácia je možná, ovšem za cenu toho, že výpočet bude pomalší. Presnejšie, na simuláciu $f_1(n)$ krokov stroja $\mathcal{M}_w$ potrebuje $\mathcal{M}$ nanajvýš $cf_1(n)\log f_1(n)$ krokov (c je konštanta závisiaca len od $\mathcal{M}_w$).

Zrejme jazyk L rozhodovaný strojom $\mathcal{M}$ patrí do $\mathsf{DTIME}(f_2(n))$. Zároveň ale jazyk L nepatrí do $\mathsf{DTIME}(f_1(n))$. □

Podobná hierarchia ako pre deterministické časové zložitostné triedy sa dá ukázať aj pre nedeterministické triedy.

Fakt 4.14 *Nech $f_1(n)$ a $f_2(n)$ sú konštruovateľné funkcie a nech $f_1(n) = o(f_2(n))$.*
Potom existuje jazyk, ktorý patrí do $\mathsf{NTIME}(f_2(n))$, ale nepatrí do $\mathsf{NTIME}(f_1(n))$.

Výsledky o hierarchiách dokazujú existenciu jazykov ľubovoľne veľkej zložitosti. Ak $L \in \mathsf{DTIME}(f_2(n)) \setminus \mathsf{DTIME}(f_1(n))$, tak hovoríme, že f_2 je *horným odhadom* časovej zložitosti jazyka L a f_1 je *dolným odhadom*. Horná hranica pre L znamená, že existuje deterministický Turingov stroj (algoritmus) A pre jazyk (problém) L s časovou zložitosťou $T_A(n) \leq f_2(n)$. Dolná hranica naopak hovorí, že neexistuje deterministický TS (algoritmus) B rozhodujúci L v čase $T_B(n) \leq f_1(n)$. Pod pojmom zložitosť jazyka pritom intuitívne chápeme zložitosť najlepšieho, asymptoticky optimálneho stroja rozhodujúceho jazyk L. Takéto chápanie však nie je správne, pretože existujú jazyky, ktoré nemajú najlepší Turingov stroj. Presnejšie je tento fakt sformulovaný v Blumovej teoréme (Blum's Speed–up Theorem). Poznamenávame len, že analogické tvrdenie platí aj pre priestor.

Veta 4.15 (Blumova teoréma) *Nech $r(n)$ je úplná rekurzívna funkcia. Potom existuje rekurzívny jazyk L taký, že pre každý deterministický Turingov stroj $\mathcal{M}_1$ akceptujúci L, existuje deterministický TS $\mathcal{M}_2$ s vlastnosťami*

(i) *Pre každý vstup w je $\mathcal{M}_1(w) = \mathcal{M}_2(w)$.*

(ii) $r(T_{\mathcal{M}_2}(n)) \leq T_{\mathcal{M}_1}(n)$ *pre skoro všetky $n \in \mathbb{N}$.*

Tvrdenie predchádzajúcej vety sa môže zdať prekvapujúce. Vychádza z nasledovnej myšlienky: Za prvé, pre každý jazyk L' existuje nekonečne veľa Turingovych strojov, ktoré ho akceptujú. Za druhé, dá sa skonštruovať jazyk L, ktorý nie je akceptovaný žiadnym Turingovym strojom malej veľkosti (s krátkym kódom), ale pre ktorý stroje väčšej a väčšej veľkosti pracujú stále efektívnejšie. Formálna konštrukcia jazyka L je založená na diagonalizácii.

Pretože Blumova teoréma môže byť aplikovaná nekonečne veľa krát, neexistuje optimálny Turingov stroj akceptujúci L. Ku každému konkrétnemu stroju $\mathcal{M}$ pre L existuje stroj $\mathcal{M}'$ pracujúci efektívnejšie než $\mathcal{M}$. Preto nemôžeme definovať zložitosť problému ako zložitosť optimálneho algoritmu riešiaceho tento problém. Namiesto toho má zmysel hovoriť o hornej a dolnej hranici zložitosti problému, čo je aj postačujúce pre klasifikáciu problémov podľa ich obtiažnosti.

Na záver ešte ukážeme výsledok, ktorý ukazuje, čo sa stane, ak v predpokladoch viet o hierarchiách vypustíme požiadavku na konštruovateľnosť funkcie $f_2(n)$.

Veta 4.16 *Existuje úplná rekurzívna funkcia $f : \mathbb{N} \to \mathbb{N}$ taká, že*

$$\mathsf{DTIME}(f(n)) = \mathsf{DTIME}(2^{f(n)}).$$

Dôkaz Definujeme funkciu f takým spôsobom, že výpočet každého deterministického Turingovho stroja na vstupe dĺžky n, sa zastaví buď po nanajvýš $f(n)$ krokoch, alebo po viac než $2^{f(n)}$ krokoch.

Uvažujme rastúce usporiadanie kódov Turingovych strojov $\mathcal{M}_0, \mathcal{M}_1, \mathcal{M}_2, \ldots$. Pre čísla $i, k \geq 0$ definujme vlastnosť $P(i,k)$:

> Výpočet každého spomedzi strojov $\mathcal{M}_0, \mathcal{M}_1, \ldots, \mathcal{M}_i$ na každom vstupe dĺžky i sa buď zastaví po nanajvýš k krokoch, alebo sa zastaví po viac než 2^k krokoch, alebo sa nezastaví vôbec.

Napriek tomu, že niektoré spomedzi uvažovaných výpočtov môžu byť nekonečné, vlastnosť $P(i,k)$ je rozhodnuteľná. Stačí pre každý zo strojov $\mathcal{M}_0$, $\mathcal{M}_1, \ldots$,$\mathcal{M}_i$ simulovať prvých 2^{k+1} krokov výpočtu na každom vstupe dĺžky i.

Definujeme hodnotu $f(i)$ pre $i \geq 0$. Označme Σ_j vstupnúnú abecedu stroja $\mathcal{M}_j$ z uvažovanej postupnosti $\mathcal{M}_0, \mathcal{M}_1, \mathcal{M}_2, \ldots$. Uvažujeme postupnosť čísiel $k_1, k-2, \ldots$, kde

$$k_1 \stackrel{\text{def}}{=} 2i \qquad\qquad k_j \stackrel{\text{def}}{=} 2^{k_{j-1}} + 1 \text{ pre } j \geq 2.$$

Nech $N(i) = \sum_{j=0}^{i} |\Sigma_j|^i$, tj. $N(i)$ je celkový počet vstupov dĺžky i pre prvých $i+1$ strojov. Každý z týchto vstupov môže spôsobiť pre nanajvýš jednu hodnotu k_j, že vlastnosť $P(i, k_j)$ nebude splnená. Preto musí existovať číslo $l \leq N(i)$ také, že $P(i, k_l)$ platí. Teraz sme pripravení definovať funkciu f. Hodnotu $f(i)$ položíme $f(i) \stackrel{\text{def}}{=} k_l$.

Nech teraz jazyk L patrí do $\mathsf{DTIME}(2^{f(n)})$. L je rozhodovaný nejakým Turingovym strojom, nazvime ho $\mathcal{M}_j$, v čase $2^{f(n)}$. Potom pre každý vstup w dĺžky $|w| \geq j$ (teda pre všetky vstupy až na konečno mnoho), nie je možné, aby sa výpočet stroja $\mathcal{M}_j$ zastavil po viac než $f(|w|)$ ale menej než $2^{f(|w|)}$ krokoch (dôvodom je fakt, že pri konštrukcii funkcie f sme brali do úvahy aj stroj $\mathcal{M}_j$). Keďže výpočet stroja $\mathcal{M}_j$ sa určite zastaví po nanajvýš $2^{f(|w|)}$ krokoch, môžeme dedukovať, že sa zastaví po nanajvýš $f(|w|)$ krokoch. Táto vlastnosť samozrejme platí len pre vstupy dĺžky aspoň j. Zostáva nám len konečne veľa slov, pre ktoré nepoznáme dĺžku výpočtu. Avšak stroj $\mathcal{M}_j$ dokážeme modifikovať tak, že vymenujeme všetky slová dĺžky menšej než j patriace do jazyka L. Týchto slov je konečne veľa a preto zmenou množiny stavov stroja $\mathcal{M}_j$ dosiahneme, že $\mathcal{M}_j$ bude rozhodovať slová dĺžky menšej než j v čase úmernom ich dĺžke.

Zhrnutím dostávame, že stroj $\mathcal{M}_j$ rozhoduje jazyk L v čase $f(n)$ a teda

$$\mathsf{DTIME}(f(n)) = \mathsf{DTIME}(2^{f(n)}).$$

$\square$

Ľahko sa dá nahliadnuť, že funkciu 2^n v predchádzajúcej vete možno nahradiť akoukoľvek inou rýchlo rastúcou funkciu. Analogickým spôsobom by sa podobný výsledok ukázal aj pre priestorovú zložitosť.

4.3 Vzťahy medzi zložitostnými triedami

Výsledky o hierarchiách nám ukazujú, ako sa mení zložitostná trieda daného typu, keď sa mení funkcia, ktorá ju určuje. Avšak omnoho zaujímavejšou otázkou je, aký je vzťah medzi zložitostnými triedami *rôznych typov*. Napriek tomu, že tento problém je jedným z najskúmanejších v teórii zložitosti, sú otázky vzťahov medzi niektorými zložitostnými triedami, napr. medzi P a NP, otvoreným problémom.

Veta 4.17 *Nech f je konštruovateľná funkcia. Potom*

a) $\mathsf{DSPACE}(f(n)) \subseteq \mathsf{NSPACE}(f(n))$ *a* $\mathsf{DTIME}(f(n)) \subseteq \mathsf{NTIME}(f(n))$

b) $\mathsf{NTIME}(f(n)) \subseteq \mathsf{DSPACE}(f(n))$

c) $\mathsf{NSPACE}(f(n)) \subseteq \bigcup_{c>0} \mathsf{DTIME}(c^{f(n)})$ *pre funkciu* $f(n) \geq \log n$

d) $\mathsf{NTIME}(f(n)) \subseteq \bigcup_{c>0} \mathsf{DTIME}(c^{f(n)})$

e) $\mathsf{NSPACE}(f(n)) \subseteq \mathsf{DSPACE}((f(n))^2)$ *pre funkciu* $f(n) \geq \log n$

Dôkaz Časť **a)** je triviálna; každý deterministický stroj je zároveň nedeterministickým.

Platnosť inklúzie **b)** vyplýva z dôkazu vety 3.19, pretože priestorová zložitosť tam navrhnutého stroja $\mathcal{D}$ je $\mathcal{O}(f(n))$. Stroj $\mathcal{D}$ postupne simuluje všetky možné výpočty nedeterministického stroja $\mathcal{N}$. Pri simulácii jedného výpočtu si uchováva iba informáciu o aktuálnej konfigurácii; na simulovanie rôznych výpočtov používa vždy ten istý priestor.

Ďalej ukážeme platnosť tvrdenia **c)**. Dôkaz tejto inklúzie tiež nie je obtiažny; využíva sa pri ňom metóda *dosiahnuteľnosti*.

Nech jazyk L je rozhodovaný presným nedeterministickým TS $\mathcal{M}$ so vstupnou páskou a s priestorovou zložitosťou f, kde f je konštruovateľná funkcia. Definujeme pojem *konfiguračného grafu* $G(\mathcal{M}, w)$ stroja $\mathcal{M}$ na vstupe w. Vrcholy orientovaného grafu $G(\mathcal{M}, w)$ sú konfigurácie, ktoré sa môžu vyskytnúť vo výpočte $\mathcal{M}$ na w. Medzi vrcholmi C_1 a C_2 je hrana práve vtedy, ak $\mathcal{M}$ môže prejsť v jednom kroku výpočtu z konfigurácie C_1 do konfigurácie C_2. Zrejme určiť, či $w \in L$, je ekvivalentné problému rozhodnúť, či v konfiguračnom grafe $G(\mathcal{M}, w)$ existuje cesta z vrcholu označeného počiatočnou konfiguráciou C_0 do vrcholu zodpovedajúceho akceptujúcej konfigurácii. Pritom pre každý vstup w môžeme počet vrcholov grafu $G(\mathcal{M}, w)$ zhora ohraničiť číslom $c^{\log|w|+f(|w|)}$ (c je konštanta, ktorej hodnota závisí len stroja $\mathcal{M}$ a nezávisí od dĺžky vstupu)[1].

Inak povedané, problém rozhodnúť, či $\mathcal{M}$ akceptuje w dĺžky n, sme redukovali na problém *dosiahnuteľnosti* v grafe s $c^{\log n+f(n)}$ vrcholmi. Preto môžeme na riešenie nášho problému využiť známy algoritmus[2] na rozhodovanie dosiahnuteľnosti v grafe, ktorého časová zložitosť je $\mathcal{O}(m^2)$ pre grafy o m vrcholoch. Časová zložitosť deterministického stroja simulujúceho nedeterministický stroj $\mathcal{M}$ je $\mathcal{O}(2^{\log n+f(n)})$, čo sme potrebovali dokázať.

[1] Pozri dôkaz lemy 4.9

[2] Algoritmus pre problém DOSIAHNUTEĽNOSŤ je neformálne popísaný v časti 2.

Naznačíme trochu podrobnejšie, akým spôsobom je možné realizovať spomínaný algoritmus na deterministickom Turigovom stroji. Existuje niekoľko možností. Prvou z nich je najprv skonštruovať incidenčnú maticu grafu $G(\mathcal{M}, w)$ pre daný vstup w a potom simulovať výpočet stroja rozhodujúceho dosiahnuteľnosť. Elegantnejší spôsob nepredpokladá konštrukciu incidenčnej matice. Namiesto toho kedykoľvek algoritmus potrebuje vedieť, či graf obsahuje hranu (C, C'), invokuje jednoduchý podprogram, ktorý overí či $\mathcal{M}$ môže v jednom kroku prejsť z konfigurácie C do C'.

Inklúzia **d)** špecifikuje vzťah medzi nedeterministickým a deterministikým časom a je obsahom vety 3.19.

Nakoniec dokážeme vzťah pre priestorovú zložitosť (tvrdenie **e)**). Tento výsledok je známy aj pod názvom *Savitchova teoréma.*

Nech jazyk L je rozhodovaný presným nedeterministickým TS $\mathcal{M}$ so vstupnou páskou a s priestorovou zložitosťou f, kde f je konštruovateľná funkcia. Navrhneme deterministický TS $\overline{\mathcal{M}}$ rozhodujúci L v priestore $\mathcal{O}((f(n))^2)$. Podobne ako pri dôkaze tvrdenia c) vyjdeme z toho, že problém rozhodnúť, či $\mathcal{M}$ akceptuje vstup w, vieme redukovať na problém dosiahnuteľnosti v konfiguračnom grafe $G(\mathcal{M}, w)$. Algoritmus, ktorý sme uvažovali v dôkaze inklúzie c), nám nevyhovuje, pretože jeho priestorová zložitosť je $2^{\mathcal{O}(f(n))}$. Navrhneme nový algoritmus, ktorý pracuje na princípe *divide et impera*.

Lema 4.18 *DOSIAHNUTEĽNOSŤ* $\in \mathsf{DSPACE}(\log^2 n)$.

Dôkaz Nech G je orientovaný graf s n vrcholmi a nech x a y sú dva jeho vrcholy. Definujeme predikát $\mathsf{CESTA}(x, y, i)$ ktorý je splnený práve ak existuje cesta dĺžky nanajvýš 2^i z x do y v G. Keďže cesta z x do y v G môže mať dĺžku nanajvýš n, tak problém nájdenia cesty z x do y je ekvivalentný problému, či predikát $\mathsf{CESTA}(x, y, \lceil \log n \rceil)$ je splnený.

Skonštruujeme Turingov stroj so vstupnou a dvoma pracovnými páskami, ktorý bude rozhodovať predikát $\mathsf{CESTA}(x, y, i)$. Ak stroj dostane ako vstup incidenčnú maticu grafu G o n vrcholoch a vrcholy x a y, zapíše (binárne) na svoju prvú pracovnú pásku čísla vrcholov x, y a číslo $\lceil \log n \rceil$. Druhú pracovnú pásku používa na pomocné výpočty; vopred na nej vyznačí $\lceil \log n \rceil$ políčok a nikdy nezapisuje mimo tejto vyznačenej oblasti.

Popíšeme spôsob, akým TS zistí, či existuje cesta z x do y v G. Ak $i = 0$, tak TS má rozhodnúť, či existuje cesta dĺžky nanajvýš $2^0 = 1$ spájajúca x a y. Preto stačí aby overil, či $x = y$ alebo či (x, y) je hranou grafu G (zistí z incidenčnej matice). Ak $i \geq 1$ tak platnosť predikátu overí aplikovaním nasledovnej rekurzívnej procedúry

Procedúra $\mathsf{CESTA}(x, y, i)$
<u>**if**</u> $i = 0$
  <u>**then**</u> <u>**if**</u> $x = y \vee x \vdash y$ <u>**then**</u> *return* true
    <u>**else**</u> <u>**for**</u> každý vrchol z grafu G <u>**do**</u>
      <u>**if**</u> $\mathsf{CESTA}(x, z, i-1) \wedge \mathsf{CESTA}(z, y, i-1)$ <u>**then**</u> *return* true
      <u>**od**</u>
      return false

Algoritmus je založený na jednoduchej úvahe, vychádzajúcej z toho, že každá cesta dĺžky nanajvýš 2^i z x do y musí mať *stred* z taký, že cesty z x do z a zo z do y majú dĺžku nanajvýš polovičnú, t.j. 2^{i-1}. Implementácia algoritmu na Turingovom stroji je jednoduchá: stroj generuje vrcholy z jeden za druhým, využívajúc ten istý priestor. Zakaždým, keď vygeneruje nové z, zapíše na koniec prvej pracovnej pásky trojicu $(x, z, i-1)$. Rekurzívne volá procedúru rozhodujúcu predikát $\mathsf{CESTA}(x, z, i-1)$. Ak dostane negatívnu odpoveď, vymaže z prvej pracovnej pásky trojicu $(x, z, i-1)$ a vygeneruje nasledujúce z. Pri pozitívnej odpovedi vymaže $(x, z, i-1)$ z prvej pracovnej pásky, zapíše na ňu novú trojicu $(z, y, i-1)$ a znovu rekurzívne volá algoritmus rozhodujúci predikát $\mathsf{CESTA}(z, y, i-1)$. Ak odpoveď je negatívna, zmaže trojicu a generuje nové z; ak odpoveď je pozitívna, predikát $\mathsf{CESTA}(x, y, i)$ je splnený. Poznamenávame, že počas celého výpočtu stroj používa prvú pracovnú pásku ako zásobník, v ktorom si ukladá informácie o jednotlivých rekurzívnych volaniach.

Zostáva určiť priestorovú zložitosť navrhnutého stroja. Na prvej pracovnej páske nikdy nebude zapísaných viac než $\lceil \log n \rceil$ trojíc. Každá trojica má dĺžku nanajvýš $3 \log n$. Priestorová zložitosť je $\mathcal{O}((\log n)^2)$. □

Pokračovanie dôkazu Savitchovej teorémy. Deterministický stroj $\overline{\mathcal{M}}$ simulujúci výpočet nedeterministického stroja $\mathcal{M}$ na slove w volá pre každú prípustnú akceptujúcu konfiguráciu Turingov stroj rozhodujúci problém DOSIAHNUTEĽNOSŤ. Jediný rozdiel je v tom, že testovanie, či skúmaná hrana (x, y) patrí do grafu sa nahradí testom, či $\mathcal{M}$ môže v jednom kroku prejsť z konfigurácie x do konfigurácie y. Pretože počet vrcholov konfiguračného grafu $G(\mathcal{M}, w)$ nepresiahne $c^{f(|w|)}$ (c je konštanta, ktorej hodnota závisí len od $\mathcal{M}$), tak priestorová zložitosť stroja $\overline{\mathcal{M}}$ je $\mathcal{O}((f(n))^2)$. □

Zhrnutím poznatkov získaných v predchádzajúcich častiach dostávame:

$$\mathsf{DLOG} \subseteq \mathsf{NLOG} \subseteq \mathsf{P} \subseteq \mathsf{NP} \subseteq \mathsf{PSPACE} \subseteq \mathsf{EXPTIME} \subseteq \mathsf{NEXPTIME}.$$

O všetkých inklúziách v prezentovanej hierarchii predpokladáme, že sú ostré. Ale ani pre jednu z nich sa to doteraz nepodarilo dokázať (ani vyvrátiť) napriek tomu, že tento problém rozhodne patrí k najviac skúmaným v teórii zložitosti. Výsledky o časovej resp. priestorovej hierarchii pritom ukazujú, že niektorá z uvedených inklúzií musí byť ostrá. Platí totiž

$$\mathsf{DLOG} \subset \mathsf{PSPACE}$$

$$\mathsf{P} \subset \mathsf{EXPTIME}$$

$$\mathsf{NP} \subset \mathsf{NEXPTIME}$$

Výsledky, ktoré sme doteraz prezentovali, samozrejme nedávajú odpoveď na vzťah ľubovoľných dvoch zložitostných tried. Ukážeme preto ešte na záver tejto časti dôkazovú techniku, ktorá môže byť v mnohých prípadoch nápomocná.

Veta 4.19 *Ak* $\mathsf{DSPACE}(n) \subseteq \mathsf{P}$ *tak* $\mathsf{P} = \mathsf{PSPACE}$.

Dôkaz Predpokladajme, že $\mathsf{DSPACE}(n) \subseteq \mathsf{P}$. Nech jazyk $L \in \mathsf{PSPACE}$. Chceme ukázať, že $L \in \mathsf{P}$.

Nech $\mathcal{M}$ je deterministický TS, s polynomiálnou priestorovou zložitosťou $p(n)$, rozhodujúci jazyk L. Skonštruujeme nový jazyk L' takto:

$$L' \stackrel{\text{def}}{=} \{\, w10^{p(|w|)} \ \mid \ w \in L \,\}.$$

Ukážeme, že $L' \in \mathsf{DSPACE}(n)$. Deterministický TS $\mathcal{M}'$ rozhodujúci jazyk L' v lineárnom priestore pracuje takto:

1. $\mathcal{M}'$ hľadá vo vstupnom slove najpravejší symbol 1
2. $\mathcal{M}'$ simuluje výpočet stroja $\mathcal{M}$ na slove, ktoré predchádza najpravejší symbol 1.

Vo výpočte na vstupe $w10^{p(|w|)}$ použije $\mathcal{M}'$ priestor $p(|w|)$, čo je priestor lineárny vzhľadom k dĺžke vstupu.

Z predpokladu $\mathsf{DSPACE}(n) \subseteq \mathsf{P}$ vyplýva existencia polynomiálne časovo ohraničeného deterministického stroja $\mathcal{M}''$ rozhodujúceho jazyk L'. Využijúc stroj $\mathcal{M}''$ skonštruujeme nový stroj $\overline{\mathcal{M}}$ rozhodujúci jazyk L v polynomiálnom čase. Stroj $\overline{\mathcal{M}}$ pracuje na vstupe w takto:

1. $\overline{\mathcal{M}}$ skopíruje vstup w na prvú pracovnú pásku a pripojí k nemu reťazec $10^{p(|w|)}$. Využije sa pritom konštruovateľnosť funkcie $p(n)$.
2. $\overline{\mathcal{M}}$ simuluje výpočet stroja $\mathcal{M}''$ na slove, ktoré má zapísané na prvej pracovnej páske.

Tým sme dokázali, že $L \in \mathsf{P}$ a teda $\mathsf{PSPACE} \subseteq \mathsf{P}$. Pretože opačná inklúzia je zrejmá, je tvrdenie vety dokázané. □

Dôsledok 4.20 $\mathsf{P} \neq \mathsf{DSPACE}(n)$

4.4 Komplement zložitostných tried

Definovanie nedeterminizmu prináša so sebou istú asymetriu: zatiaľ čo na akceptovanie vstupu stačí *existencia jedného* akceptujúceho výpočtu, na to aby sme zamietli vstup, musíme overiť, či *všetky* výpočty sú zamietajúce. V tejto časti ukážeme, aké dôsledky má táto asymetria na komplementy zložitostných tried.

Nech $L \subseteq \Sigma^*$ je jazyk. *Komplement* jazyka L, označujeme ho $\overline{L}$ alebo co–L, je jazyk

$$\mathsf{co}\text{–}L \stackrel{\text{def}}{=} \Sigma^* \setminus L.$$

Túto definíciu môžeme rozšíriť aj na rozhodovacie problémy: komplement rozhodovacieho problému A (zvyčajne používame označenia A KOMPL) je problém, ktorého riešenie je „áno" kedykoľvek je riešenie toho istého prípadu problému A „nie" a naopak. Napríklad DOSIAHNUTEĽNOSŤ KOMPL je problém, ktorého riešením pre daný graf G a jeho dva vrcholy x a y je „áno" práve vtedy, ak neexistuje cesta z x do y v G. Formálne vzaté, jazyky zodpovedajúce problémom DOSIAHNUTEĽNOSŤ a DOSIAHNUTEĽNOSŤ KOMPL nie sú komplementárne, pretože ich zjednotenie netvorí množinu Σ^*, ale množinu slov, správne kódujúcich grafy (s dvoma špecifikovanými vrcholmi).

Pre každú zložitostnú triedu $\mathcal{C}$ definujeme jej komplement co–$\mathcal{C}$ ako triedu

$$\mathsf{co}\text{–}\mathcal{C} \stackrel{\text{def}}{=} \{\, \mathsf{co}\text{–}L \mid L \in \mathcal{C} \,\}.$$

Zrejme deterministické časové aj priestorové zložitostné triedy sú uzavreté na komplement.Každý deterministický TS rozhodujúci jazyk L, môžeme transformovať na stroj rozhodujuci co–L tak, že navzájom zameníme stavy q_{accept} a q_{reject}Presnejšie, ak f je konštruovateľná funkcia, tak

$$\mathsf{DTIME}(f(n)) = \mathsf{co}\text{–}\mathsf{DTIME}(f(n))$$
$$\mathsf{DSPACE}(f(n)) = \mathsf{co}\text{–}\mathsf{DSPACE}(f(n)).$$

Analogický vzťah platí aj pre nedeterministické priestorové zložitostné triedy. Dôkaz tohoto tvrdenia, ako ukážeme v nasledujúcej vete, nie je až tak priamočiary. Otázka, či nedeterministické časové zložitostné triedy sú uzavreté na komplement je dôležitým otvoreným problémom, známym predovšetkým vo formulácii $\mathsf{P} \stackrel{?}{=} \mathsf{co}\text{–}\mathsf{NP}$.

Vráťme sa teraz späť k otázke komplementu nedeterministických priestorovo ohraničených zložitostných tried. Tento problém bol prvýkrát sformulovaný začiatkom 60-tych rokov, ale vyriešili ho až v roku 1987 nezávisle na sebe Neil Immerman a Róbert Szelepcsényi. Kľúčom k riešeniu bolo nájdenie $\log n$–priestorovo ohraničeného Turingovho stroja, ktorý rieši rozšírenú verziu problému DOSIAHNUTEĽNOSŤ: stroj určí počet vrcholov, ktoré sú dosiahnuteľné z daného vrcholu x.

Lema 4.21 *Pre daný orientovaný n–vrcholový graf G a vrchol x sa počet vrcholov dosiahnuteľných v G z x dá vypočítať nedeterministickým Turingovym strojom v priestore* $\log n$.

Dôkaz Podobne ako v predchádzajúcich dôkazoch, najprv vysvetlíme algoritmus na riešenie daného problému a potom špecifikujeme jeho implementáciu na nedeterministickom Turingovom stroji.

Algoritmus pozostáva zo štyroch vnorených cyklov. Každý z cyklov je pomerne jednoduchý, ale pretože využívajú nedeterminizmus, ich vzájomná kombinácia nie je triviálna.

Vonkajší cyklus počíta iteratívne $|S(1)|, |S(2)|, \ldots, |S(n-1)|$, kde $S(k)$ je množina vrcholov grafu G, ktoré sa dajú dosiahnuť z vrcholu x po ceste dĺžky nanajvýš k. Zrejme ak n je počet vrcholov grafu, tak $|S(n-1)|$ je hľadaným riešením nášho problému. Vždy, keď sa vypočíta $|S(k-1)|$, začína výpočet hodnoty $|S(k)|$. Na začiatku vieme, že $|S(0)| = 1$. Takže vonkajší cyklus je

$|S(0)| := 1$;
for $k = 1, 2, \ldots, n-1$ **do**
  vypočítaj $|S(k)|$ *z* $|S(k-1)|$
od

Pri výpočte hodnoty $|S(k)|$ využívame $|S(k-1)|$, (ale, našťastie, žiadne iné z predchádzajúcich $|S(j)|$). Počítadlo l sa nastaví na nulu. Potom preverujeme všetky vrcholy grafu G, jeden po druhom, v číselnom usporiadaní, využívajúc ten istý priestor. Ak nájdeme vrchol, ktorý patrí do $S(k)$, zvýšime počítadlo l o jednotku. Na konci obsahuje počítadlo l práve hodnotu $|S(k)|$:

$l := 0$
for *každý vrchol* $u = 1, 2, \ldots, n$ **do**
    if $u \in S(k)$ **then** $l := l + 1$
      od

Zatiaľ sme ale nepovedali, ako rozhodneme, či $u \in S(k)$. Určíme to pomocou tretieho cyklu. V tomto cykle budeme opäť prechádzať cez všetky vrcholy grafu G v číselnom usporiadaní, využívajúc ten istý priestor. Ak vrchol v je z množiny $S(k-1)$, tak zvýšime počítadlo m o jednotku. V tomto počítadle si pamätáme počet už nájdených prvkov množiny $S(k-1)$. Potom overíme, či $u = v$ alebo či existuje v G hrana z v do u (v ďalšom bude $G(v, u)$ označovať, že buď $u = v$ alebo existuje hrana z v do u). Ak áno, našli sme u patriace do $S(k)$; nastavíme hodnotu premennej *odpoved'* na true. Ak sa nám nepodarilo ukázať, že $u \in S(k)$, tak hodnotou premennej *odpoved'* bude false. *Ak ale porovnaním vypočítanej hodnoty m a známeho čísla* $|S(k-1)|$ *zistíme, že sme nevzali do úvahy všetky prvky množiny* $S(k-1)$, (a takáto situácia môže nastať, pretože o vzťahu $v \in S(k-1)$ rozhodujeme nedeterministicky), *tak výpočet ukončíme v zamietajúcom stave* . Tento výpočet neovplyvní výsledok.

$m := 0$;
odpoved' := false;
for *každý vrchol* $v = 1, 2, \ldots, n$ **do**
    if $v \in S(k-1)$ **then** $m := m + 1$;
            if $G(v, u)$ **then** *odpoved'* := true
              od;
              if $m < |S(k-1)|$ **then** *koniec výpočtu; vstup zamietnutý*
                else if *odpoved'* **then** $u \in S(k)$
                  else $u \notin S(k)$

Stále nám ale zostáva ukázať, ako určíme, či $v \in S(k-1)$. Platnosť tejto podmienky overíme *nedeterministicky* a to tak, že nedeterministicky uhádneme $k-2$ vrcholov $w_1, \ldots, w_{k-2}$ a otestujeme, či vrcholy $x, w_1, \ldots, w_{k-1}, v$ tvoria cestu v grafe G. Pritom nevylučujeme možnosť, že $w_i = w_j$ pre nejaké i, j, pretože potrebujeme overiť existenciu cesty dĺžky *nanajvýš* $k-1$.

$w_0 := x$;
for $p = 1, \ldots, k-1$ **do**
    uhádni vrchol w_p;
    if $\neg G(w_{p-1}, w_p)$ **then** *koniec výpočtu; vstup zamietnutý*
      od;
      if $w_{k-1} = v$ **then** $v \in S(k-1)$
        else *koniec výpočtu; vstup zamietnutý*

Tým sme ukončili popis algoritmu. Algoritmus sa dá realizovať nedeterministickým $\log n$–priestorovo ohraničeným Turingovym strojom $\mathcal{M}$. $\mathcal{M}$ si v priebehu výpočtu uchováva na separátnych páskach hodnoty deviatich premenných $k, |S(k-1)|, l, u, m, v, p, w_p$ a w_{p-1} plus vstupný a výstupný reťazec. Hodnoty všetkých premenných sa počas výpočtu iba zvyšujú o jednotku, sú porovnávané navzájom a so vstupom. Hodnota žiadnej premennej neprevýši n.

Dokážeme korektnosť navrhnutého algoritmu, t.j. že správne vypočíta hodnoty $|S(k)|$ pre všetky k. Tvrdenie platí pre $k = 0$. Uvažujme úspešne vypočítanú hodnotu $|S(k)|$ (t.j. vypočítanú vo výpočte, v ktorom nikdy nedošlo k situácii $m < |S(k-1)|$). Ukážeme, že obsah počítadla l sa zvýšil práve vtedy, keď skúmané v patrilo do $S(k-1)$. Keďže m–cyklus a v–cyklus boli úspešne ukončené, muselo byť m rovné $|S(k-1)|$. To znamená, že všetky $v \in S(k-1)$ boli rozpoznané, lebo najvnútornejší cyklus nikdy nepotvrdí existenciu cesty z x do v ak takáto v grafe nie je. Preto hodnota premennej *odpoved'* správne určuje, či $u \in S(k)$; ale na základe hodnoty tejto premennej sa zvyšuje hodnota počítadla l. Nakoniec, ľahko možno vidieť, že vždy existuje aspoň jeden úspešný výpočet (ten, v ktorom boli správne uhádnuté všetky prvky množiny $S(k-1)$ a cesty vedúce do nich). □

Bezprostredným dôsledkom práve dokázanej Lemy je

Veta 4.22 (Immermanova—Szelepcsényiho teoréma) *Ak f je konštruovateľná funkcia, $f(n) \geq \log n$, tak*

$$\mathsf{NSPACE}(f(n)) = \mathsf{co\text{–}NSPACE}(f(n)).$$

Dôkaz Predpokladajme, že jazyk $L \in \mathsf{NSPACE}(f(n))$ je rozhodovaný $f(n)$–priestorovo ohraničeným nedeterministickým Turingovym strojom $\mathcal{M}$. Skonštruujeme $f(n)$–priestorovo ohraničený deterministický Turingov stroj $\overline{\mathcal{M}}$ rozhodujúci $\overline{L}$. Pre vstup w, $\overline{\mathcal{M}}$ simuluje algoritmus z lemy 4.21 pre konfiguračný graf stroja $\mathcal{M}$ na vstupe w. Tak ako v predchádzajúcich aplikáciach, kedykoľvek potrebuje algorimus vedieť, či v grafe existuje daná hrana, $\overline{\mathcal{M}}$ zistí túto informáciu z prechodovej funkcie stroja $\mathcal{M}$. Kedykoľvek stroj $\overline{\mathcal{M}}$ zistí, že akceptujúca konfigurácia u patrí do množiny $S(k)$ (pre ľubovoľné k), tak ukončí výpočet a zamietne vstup. V opačnom prípade, ak bola vypočítaná hodnota $|S(k-1)|$ a žiadna akceptujúca konfigurácia nebola nájdená, tak $\overline{\mathcal{M}}$ akceptuje vstup. □

Immermanova—Szelepcsényiho teoréma dovoľuje jednoduché odvodenie hierarchie nedeterministických priestorovo ohraničených zložitostných tried. Pretože tieto triedy sú uzavreté na komplement, môžeme použiť priamu diagonalizáciu (čo predtým nebolo možné — pozri Vetu 4.12).

Časť II

Vzťah tried P a NP

Kapitola 5

Redukcie problémov a úplnosť

V predchádzajúcich častiach sme už naznačili, že poznáme len veľmi málo techník na určovanie presnej obtiažnosti problémov. Preto Cook ako prvý použil metódu, umožňujúcu získavať tzv. relatívne dolné odhady, vďaka ktorým môžeme porovnať obtiažnosť dvoch problémov aj v prípade, keď ich presnú zložitosť nepoznáme. Metóda je založená na myšlienke redukcie problémov.

Prvé, čo potrebujeme, je popísať presný význam tvrdenia, že daný problém je *aspoň tak ťažký* ako iný problém. Tento vzťah vyjadríme pomocou *redukcie*. Problém A je aspoň tak ťažký (zložitý) ako problém B, ak B sa dá redukovať na A. Neformálne, redukcia problému B na problém A znamená, že existuje transformácia R, ktorá každý prípad x problému B prevedie na ekvivalentný prípad $R(x)$ problému A. Ekvivalentý tu znamená, že riešenie prípadu $R(x)$ problému A je „áno“ práve vtedy ak riešenie prípadu x problému B je „áno“.

Možnosť redukcie problému B na problém A zodpovedá tomu, že A je aspoň tak ťažký ako B iba za predpokladu, že transformácia R nie je príliš komplikovaná. Ak neohraničíme zložitosť transformácie, môžeme sa dostať k takým absurditám, ako je redukcia problému obchodného cestujúceho (rozhodovacej verzie) POC(R) na problém DOSIAHNUTEĽNOSŤ spolu s dôsledkom, že POC(R) je ťažší problém než DOSIAHNUTEĽNOSŤ. Skutočne, stačí uvážiť takúto transformáciu: pre daný prípad x problému POC(R) preskúmaj všetky cesty; ak niektorá z nich je kratšia než dané d, tak $R(x)$ je graf s dvoma vrholmi 1 a 2 a jedinou hranou vedúcou z 1 do 2. V opačnom prípade je $R(x)$ graf s vrcholmi 1 a 2 a bez hrany. Problém je v tom, že takáto transformácia si vyžaduje exponenciálny čas.

Druhým našim cieľom bude charakterizovať v danej zložitostnej triede problémy, ktoré sú v nej relatívne najťažšie. Tieto problémy istým spôsobom reprezentujú uvažovanú zložitostnú triedu; na ich riešenie je potrebné skutočne využiť všetky zdroje (čas resp. priestor), ktoré sú v rámci danej zložitostnej triedy k dispozícii. Formalizáciou idey reprezentanta triedy je pojem úplnosti.

Pojem redukcie problémov samozrejme nie je významný len pri porovnávaní obtiažnosti problémov resp. pri hľadaní najťažších problémov v danej zložitostnej triede, ale je aj významnou metodológiou návrhu algoritmov. Na riešenie nového problému môžeme využiť algoritmus pre už vyriešený problém; stačí nájsť vhodnú redukciu prevádzajúcu prípady nového problému na prípady vyriešeného problému.

5.1 Redukovateľnosť

Definícia 5.1 *Jazyk $L_1 \subseteq \Sigma_1^*$ je* redukovateľný *na jazyk $L_2 \subseteq \Sigma_2^*$ (značíme $L_1 \leq L_2$), práve ak existuje funkcia $R : \Sigma_1^* \to \Sigma_2^*$ vypočítateľná deterministickým Turingovym strojom s polynomiálnoun časovou zložitosťou a taká, že*

$$w \in L_1 \iff R(w) \in L_2.$$

R sa nazýva redukcia *z L_1 do L_2.*
Ak $L_1 \leq L_2$ a súčasne $L_2 \leq L_1$, tak jazyky L_1 a L_2 nazveme polynomiálne ekvivalentné *(označujeme $L_1 \equiv L_2$).*

Práve definovaná redukcia sa nazýva tiež *polynomiálna redukcia*, pretože sa dá vypočítať v polynomiálnom čase.

Nasledujúce príklady ilustrujú pojem redukovateľnosti. Konkrétne, skonštruujeme redukcie medzi problémami SAT, CNF, HAM a k–KLIKA, ktoré sú formulované nasledovne:

PROBLÉM SAT
VSTUP: Booleovská formula F.
OTÁZKA: Je formula F splniteľná?

Špeciálne, ak uvažujeme len splniteľnosť booleovských formúl v konjunktívnom normálnom tvare, hovoríme o probléme CNF.

PROBLÉM HAM
VSTUP: Orientovaný graf G.
OTÁZKA: Existuje cesta v G, ktorá navštívi každý vrchol grafu práve raz?

Lema 5.2 *HAM $\leq$ CNF.*

Dôkaz Problém splniteľnosti booleovskej formuly v konjunktívnom normálnom tvare je pravdepodobne najznámejším problémom patriacim do triedy NP. Navrhneme redukciu problému nájdenia hamiltonovskej cesty v grafe na problém CNF.

Predpokladajme, že máme daný graf G. Zostrojíme booleovskú formulu $R(G)$ takú, že $R(G)$ je splniteľná práve vtedy, ak graf G má hamiltonovskú cestu. Nech G má n vrcholov $1, 2, \ldots, n$. Potom $R(G)$ bude mať n^2 booleovských premenných x_{ij} , $1 \leq i, j \leq n$. Neformálne, premenná x_{ij} reprezentuje fakt, či vrchol j je alebo nie je i–tym vrcholom hamiltonovskej cesty. $R(G)$ bude v konjuktívnom normálnom tvare a preto stačí opísať jej klauzuly. Klauzuly musia garantovať splnenie všetkých požiadaviek potrebných na to, aby hodnoty premenných x_{ij} kódovali hamiltonovskú cestu. Za prvé, vrchol j sa musí objaviť na ceste. To zachytíme klauzulou $(x_{1j} \vee x_{2j} \vee \ldots \vee x_{nj})$; takúto klauzulu budeme mať pre každé j (pre každý vrchol). Ale vrchol j sa nemôže na ceste objaviť via ckrát; vyjadríme to klauzulou $(\neg x_{ij} \vee \neg x_{kj})$. Takúto klauzulu budeme mať pre všetky j a $i \neq k$. Ďalej, aspoň jeden vrchol musí byť i–ty v poradí, čo vyjadruje klauzula $(x_{i1} \vee x_{i2} \vee \ldots \vee x_{in})$ pre každé i. Zároveň nemôžu byť súčasne dva vrcholy i–te, preto potrebujeme klauzulu $(\neg x_{ij} \vee \neg x_{ik})$ pre každé i a všetky $j \neq k$. Nakoniec, pre každú dvojicu (i, j) ktorá *nie* je hranou grafu G nesmie platiť, že j nasleduje bezprostredne po i v hamiltonovskej ceste. Preto pre každé takéto (i, j) a $k = 1, \ldots, n-1$ pridáme klauzulu $(\neg x_{ki} \vee \neg x_{k+1,j})$. Tým sme ukončili konštrukciu. Formula $R(G)$ je konjunkciou všetkých uvedených klauzulí.

Tvrdíme, že R je redukciou z HAM na CNF. Aby sme toto tvrdenie dokázali, musíme ukázať dve veci: pre každý graf G formula $R(G)$ je splniteľná vtedy a len vtedy, ak G má hamiltonovskú cestu a R je vypočítateľná v polynomiálnom čase.

Predpokladajme, že $R(G)$ je splniteľná a že ju spĺňa priradenie T. Keďže T spĺňa všetky klauzuly formuly $R(G)$, tak musí platiť, že pre každé j existuje jediné i také, že $T(x_{ij}) = 1$; inak by klauzuly $(x_{1j} \vee x_{2j} \vee \ldots \vee x_{nj})$ a $(\neg x_{ij} \vee \neg x_{kj})$ nemohli byť splnené. Podobne, klauzuly $(x_{i1} \vee x_{i2} \vee \ldots \vee x_{in})$ a $(\neg x_{ij} \vee \neg x_{ik})$ zaručujú, že pre každé i existuje jediné j také, že $T(x_{ij}) = 1$. Z toho plynie, že T skutočne reprezentuje permutáciu $\pi(1), \ldots, \pi(n)$ vrcholov grafu G, kde $\pi(i) = j$ práve ak $T(x_{ij}) = 1$. Navyše, klauzula $(\neg x_{ki} \vee \neg x_{k+1,j})$, kde (i, j) nie je hranou grafu G a $k = 1, \ldots, n-1$ zaručuje, že pre každé k, $(\pi(k), \pi(k+1))$ je hranou grafu G. Preto $\pi(1), \ldots, \pi(n)$ je hamiltonovská cesta v G.

Naopak, predpokladajme, že graf G má hamiltonovskú cestu $\pi(1), \ldots, \pi(n)$, kde π je permutácia množiny vrcholov. Potom zrejme priradenie hodnôt $T(x_{ij}) = 1$ ak $\pi(i) = j$ a $T(x_{ij}) = 0$ ak $\pi(i) \neq j$ spĺňa formulu $R(G)$.

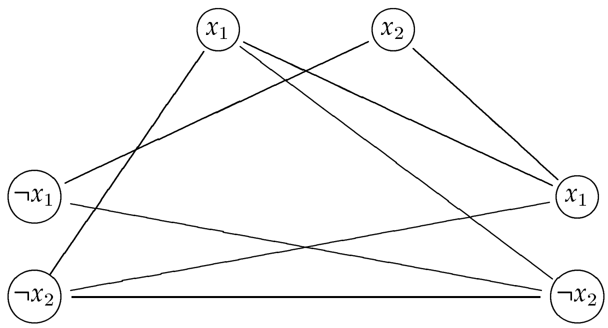
Zostáva ukázať, že R sa dá vypočítať v polynomiálnom čase. Turingov stroj počítajúci R pracuje pre vstup G takto. Najprv určí počet vrcholov n. V závislosti od n vygeneruje tie klauzuly, ktorých tvar nezávisí od tvaru grafu, ale len od počtu jeho vrcholov (prvé štyri skupiny v popise $R(G)$). Pre poslednú skupinu, Turingov stroj generuje postupne všetky klauzuly tvaru $(\neg x_{ki} \vee \neg x_{k+1,j})$ pre $k = 1, \ldots, n-1$. Pre každú z nich overí, či (i, j) je hranou grafu G; ak nie, pripojí klauzulu k výstupu. Keďže počet klauzulí ako aj ich dĺžka je polynomiálna vzhľadom k počtu vrcholov grafu G, bude mať aj výpočet Turingovho stroja polynomiálny počet krokov. $\square$

Nasledujúci príklad ukazuje, aký je vzťah medzi problémami CNF a k–KLIKA. Problém k–KLIKA je definovaný nasledovne:

PROBLÉM k–KLIKA
VSTUP: Neorientovaný graf G a konštanta $k \in \mathbb{N}$.
OTÁZKA: Obsahuje G kompletný podgraf o k vrcholoch (tzv. k–kliku)?

Lema 5.3 *CNF $\leq k$–KLIKA.*

Dôkaz Máme danú formulu F v konjunktívnom normálnom tvare. Skonštruujeme graf $R(F) = G$ a číslo k také, že F je splniteľná, práve ak G obsahuje kompletný podgraf o k vrcholoch. Číslo k volíme rovné počtu klauzúl formuly F. Vrcholy grafu G zodpovedajú všetkým výskytom literálov vo formule F. Hrana v G medzi dvoma vrcholmi, zodpovedajúcimi dvom literálom, existuje práve ak tieto literály sa nachádzajú v rôznych klauzulách a nie sú komplementárne. Napríklad, formule $(x_1 \vee x_2) \wedge (\neg x_1 \vee \neg x_2) \wedge (x_1 \vee \neg x_2)$ priradíme graf



Ukážeme, že navrhnuté zobrazenie je skutočne redukciou problému CNF na problém k–KLIKA. Vychádzame z toho, že ak má byť formula v konjunktívnom normálnom tvare splnená, musí byť v každej klauzule splnený aspoň jeden literál. Hrana v grafe medzi dvoma výskytmi literálov zodpovedá možnosti priradiť obom literálom pravdivostnú hodnotu true bez nebezpečenstva vzniku lokálneho konfliktu. Kompletný podgraf o k vrcholoch potom reprezentuje možnosť priradiť hodnotu true aspoň jednému literálu z každej klauzuly bez vzniku globálneho konfliktu. V našom príklade $k = 3$ a graf obsahuje dve 3–kliky, zodpovedajúce dvom spĺňajúcim priradeniam hodnôt.

Predpokladajme, že F je splniteľná. Nech $H : \{x_1, \dots, x_n\} \to \{\mathsf{true}, \mathsf{false}\}$ je priradenie, ktoré spĺňa F. Pri tomto priradení aspoň jeden literál z každej klauzuly musí mať hodnotu 1. Vyberme jeden takýto literál z každej klauzuly. Vrcholy grafu G zodpovedajúce týmto výskytom literálov sú všetky navzájom spojené hranami, pretože žiadne dva z nich nie sú komplementárne.

Naopak, nech G obsahuje k–kliku. Pretože graf G je k–partitný a rozdelenie grafu na k partií zodpovedá klauzulám, tak k–klika obsahuje práve jeden literál z každej klauzuly. Priraďme hodnotu true každému literálu, ktorý zodpovedá vrcholu k–kliky. Môžeme tak urobiť bez nebezpečenstva vzniku konfliktu, pretože v klike sa nemôžu objaviť žiadne dva komplementárne literály. □

Aby sme úplne špecifikovali vzťah medzi problémami CNF a k–KLIKA, ukážeme aj opačnú redukciu k–KLIKA $\leq$ CNF. Oba problémy sú teda polynomiálne ekvivalentné.

Lema 5.4 *k–KLIKA $\leq$ CNF.*

Dôkaz Nech je daný graf $G = (V, H)$ a konštanta k. Zostrojíme formulu $R(G)$ v konjunktívnom normálnom tvare, ktorá je splniteľná práve ak G obsahuje úplný podgraf o k vrcholoch.

Formula $R(G)$ má premenné x_i^u pre každé $u \in V$ a $1 \leq i \leq k$. Intuitívne, x_i^u hovorí, že u je i–tym prvkom kliky. Formula $R(G)$ je konjunkciou troch formúl C, D a E, ktorých význam a definície sú nasledovné.

- Pre každé i, $1 \leq i \leq k$, existuje aspoň jeden vrchol $u \in V$ taký, že u je i–tym prvkom kliky

$$C = \bigwedge_{i=1}^{k} \Big(\bigvee_{u \in V} x_i^u \Big).$$

- Pre každé i, $1 \leq i \leq k$, žiadne dva vrcholy nie sú súčasne i–tym prvkom kliky

$$D = \bigwedge_{i=1}^{k} (\bigvee_{\substack{u,v \in V \\ u \neq v}} (\neg x_i^u \vee \neg x_i^v).$$

- Ak (u, v) patrí do kliky, tak (u, v) je hranou grafu G. Ekvivalentne, ak (u, v) nie je hranou grafu, buď u alebo v nepatrí do kliky

$$E = \bigwedge_{(u,v) \notin H} \bigwedge_{1 \leq i,j \leq k} (\neg x_i^u \vee \neg x_j^v).$$

□

Ani pre jeden zo skúmaných problémov (CNF–SAT, k–KLIKA) nepoznáme lepší než exponenciálny algoritmus. Z práve dokázaných redukcií však vyplýva, že v okamihu, keď niekto navrhne polynomiálny algoritmus pre ktorýkoľvek z nich, máme súčasne algoritmus aj na riešenie druhého problému. Platí totiž nasledujúci vzťah:

Veta 5.5 *Ak $A \leq B$ a problém $B \in \mathsf{P}$, tak aj $A \in \mathsf{P}$.*

Dôkaz Nech $p(n)$ (p polynóm) je časová zložitosť Turingovho stroja $\mathcal{M}_B$ rozhodujúceho jazyk B a nech $r(n)$ (r polynóm) je časová zložitosť stroja $\mathcal{M}_R$ počítajúceho redukciu R. Stroj $\mathcal{M}_A$, rozhodujúci jazyk A, pre vstup w najprv simuluje $\mathcal{M}_R$ a vypočíta $R(w)$. Dĺžka slova $R(w)$ je $|R(w)| = r(|w|)$. Ďalej $\mathcal{M}_A$ simuluje stroj $\mathcal{M}_B$ na vstupe $R(w)$. Časová zložitosť simulácie je $p(r(|w|))$ a teda polynomiálna (zložením dvoch polynomiálnych funkcií získame polynomiálnu funkciu). □

Ostáva nám ešte zodpovedať nasledovnú otázku. Ak problém A je redukovateľný na problém B a B je redukovateľný na problém C: existuje redukcia A na C?

Veta 5.6 (Tranzitívnosť redukcie) *Ak R je redukcia jazyka L_1 na jazyk L_2 a R' je redukcia jazyka L_2 na jazyk L_3, tak kompozícia $R \cdot R'$ je redukciou L_1 na L_3.*

Dôkaz Platnosť ekvivalencie $w \in L_1$ práve ak $R'(R(w)) \in L_3$ vyplýva priamo z predpokladu, že R a R' sú redukcie.

Ostáva ukázať, že kompozícia $R \cdot R'$ je vypočítateľná v polynomiálnom čase. Turingov stroj počítajúci pre vstup w hodnotu $R'(R(w))$ najprv simuluje stroj počítajúci redukciu R. Jeho výstupom je $R(w)$, pričom $|R(w)| \leq p(|w|)$, kde p je polynóm. V druhej časti simuluje výpočet stroja počítajúceho redukciu R' na vstupe $R(w)$. Časová náročnosť druhej simulácie je nanajvýš $q(p(|w|)$, kde q je polynóm. Preto stroj počítajúci redukciu $R \cdot R'$ má polynomiálnu časovú zložitosť. □

5.2 Úplnosť

Keďže redukovateľnosť je tranzitívna, poskytuje čiastočné usporiadanie problémov podľa ich obtiažnosti. Nás budú špeciálne zaujímať najťažšie problémy v tomto usporiadaní, tzv. *úplné* problémy.

Definícia 5.7 *Nech $\mathcal{C}$ je zložitostná trieda.*
Jazyk L nazveme $\mathcal{C}$–ťažký, *ak každý jazyk $L' \in \mathcal{C}$ sa dá redukovať na L.*
Jazyk L nazveme $\mathcal{C}$–úplný *ak je $\mathcal{C}$–ťažký a naviac patrí do $\mathcal{C}$.*

Rozdiel medzi oboma prezentovanými pojmami je v tom, že ak jazyk L je $\mathcal{C}$–ťažký, tak nemusí patriť do triedy $\mathcal{C}$. V skutočnosti jeho zložitosť môže byť oveľa vyššia a teda nie je vhodným reprezentantom triedy $\mathcal{C}$. Pojem úplných problémov je jedným z centrálnych pojmov teórie zložitosti. Na jednej strane je to dané tým, že skutočnosť, že skúmaný problém je úplným pre určitú zložitostnú triedu, plne charaktrerizuje jeho náročnosť. Z druhej strany, úplné problémy predstavujú vždy tie najťažšie problémy z danej zložitostnej triedy. Je pre ne najmenej pravdepodobné, že patria do menšej zložitostnej triedy. Presná formulácia vlastností úplných a ťažkých problémov je obsiahnutá v leme 5.9.

Definícia 5.8 *Nech $\mathcal{C}$ je zložitostná trieda a L jazyk patriaci do $\mathcal{C}$. Trieda $\mathcal{C}$ je* uzavretá vzhľadom na redukciu *práve ak pre každý jazyk $\overline{L}$ platí*

$$\overline{L} \leq L \Longrightarrow \overline{L} \in \mathcal{C}.$$

Lema 5.9 *Nech $\mathcal{C}, \mathcal{C}'$ sú zložitostné triedy, $\mathcal{C}' \subseteq \mathcal{C}$.*

1. *Ak A je $\mathcal{C}$–úplný, $A \leq B$ a $B \in \mathcal{C}$, tak aj B je $\mathcal{C}$–úplný.*
2. *Ak A je $\mathcal{C}$–úplný, tak* co*–A je* co*–$\mathcal{C}$–úplný.*
3. *Ak A je $\mathcal{C}$–ťažký, trieda $\mathcal{C}'$ je uzavretá vzhľadom na redukciu a $A \in \mathcal{C}'$, tak $\mathcal{C}' = \mathcal{C}$.*

Dôkaz Vyplýva priamo z definície úplnosti a uzavretosti na redukciu. □

Ľahko overíme, že napr. triedy P, NP a PSPACE sú uzavreté vzhľadom na redukciu. Preto ak o ľubovoľnom NP–úplnom probléme L sa ukáže, že patrí do triedy P, tak bezprostredným dôsledkom je kolapsia tried NP a P (t.j. NP = P). Analogicky napríklad príslušnosť PSPACE–úplného problému do triedy P znamená rovnosť tried P a PSPACE, atď. Naopak, ak P $\subset$ NP, tak každý NP–úplný jazyk patrí do NP − P (a teda ho považujeme za prakticky neriešiteľný). Pojem úplných problémov je tak užitočným nástrojom na skúmanie vzťahu medzi zložitostnými triedami a na odhadovanie zložitosti konkrétnych problémov.

5.2.1 NP–úplné problémy

Pojem NP–úplnosti má špeciálny teoretický aj praktický význam. Dôkaz, že konkrétny problém je NP–úplný, má tri bezprostredné dôsledky. Za prvé, že v súčastnosti nie je známy žiaden polynomiálny algoritmus na riešenie tohoto problému—všetky známe algoritmy sú založené na úplnom prehľadávaní všetkých (exponenciálne veľa) možností. To je nepríjemný dôsledok vzhľadom k tomu, že mnoho prakticky dôležitých problémov je NP–úplných. Druhý, príjemnejší, dôsledok je, že stále existuje šanca, že pre uvažovaný problém sa polynomiálny algoritmus niekomu podarí skonštruovať. Inými slovami, nie je vylúčená existencia polynomiálneho algoritmu pre uvažovaný problém. Tretím, najpríjemnejším, dôsledkom je, že ak sa nám podarí skonštruovať polynomiálny algoritmus pre jeden NP–úplný problém, tak ľahko navrhneme polynomiálne algoritmy pre všetky NP–úplné problémy.

Poznáme dve základné metódy na dôkaz NP–úplnosti jazyka L.

Priama metóda Ukážeme, že $L \in$ NP. Táto časť dôkazu býva obvykle jednoduchá. Následne ukážeme, že každý jazyk z NP sa dá v polynomiálnom čase redukovať na L. Presnejšie to znamená pre každý nedeterminisitický Turingov stroj $\mathcal{M}$ pracujúci v polynomiálnom čase a pre každý vstup x stroja $\mathcal{M}$ skonštruovať v polynomiálnom čase slovo $y_{\mathcal{M},x}$ také, že $y_{\mathcal{M},x} \in L$ vtedy a len vtedy ak $\mathcal{M}$ akceptuje x.

Redukčná metóda Ukážeme, že $L \in$ NP a že $\overline{L} \leq L$ pre NP–úplný jazyk $\overline{L}$. NP–úplnosť jazyka L plynie z tranzitivity relácie $\leq$.

Priamy dôkaz NP–úplnosti problému splniteľnosti

V tejto časti ukážeme použitím priamej metódy, že problém splniteľnosti booleovskej formuly v konjunktívnom normálnom tvare je NP–úplný.

Veta 5.10 *Problém CNF je* NP*–úplný.*

Dôkaz Príslušnosť jazyka do NP je zrejmá—stroj (nedeterministicky) priradí hodnoty premenným a (deterministicky) overí, či sa jedná o spĺňajúce priradenie. V pozitívnom prípade vstup akceptuje, inak zamietne.

Nech $L \subseteq \Sigma^*$ je ľubovoľný, ale fixovaný jazyk z NP. Potom L je rozhodovaný nejakým nedeterministickým Turingovym strojom $\mathcal{M}$ s polynomiálnou časovou zložitosťou $p(n)$; môžeme predpokladať, že $\mathcal{M}$ je jednopáskový. Definujeme redukciu R, ktorá ľubovoľnému slovu $w \in \Sigma^*$ priradí booleovskú formulu $R(w) = F$ v konjunktívnom normálnom tvare takú, že F je splniteľná práve vtedy ak $\mathcal{M}$ akceptuje w. Pritom redukcia R je vypočítateľná deterministicky v polynomiálnom čase a jej definícia závisí len od $\mathcal{M}$.

Pri konštrukcii R vyjdeme z nasledovnej úvahy. Každý jednotlivý výpočet stroja $\mathcal{M}$ na fixovanom vstupe w je postupnosť konfigurácií, ktoré za sebou nasledujú v súlade s prechodovou reláciou δ. Pritom dĺžka výpočtu je nanajvýš $p(|w|) = N$. Ak pripustíme, že v postupnosti sa tá istá konfigurácia môže opakovať viackrát za sebou, tak každý výpočet stroja $\mathcal{M}$ na w môžeme prezentovať ako postupnosť práve $N + 1$ konfigurácií. Naviac, vo výpočte dĺžky N nemôže stroj $\mathcal{M}$ zapísať viac než N políčok. Preto obsah pásky v každej konfigurácii uvažovanej postupnosti môžeme jendoznačne reprezentovať ako reťazec dĺžky N.

V ďaľšom sa budeme snažiť zakódovať výpočet stroja $\mathcal{M}$ na w pomocou pravdivostných hodnôt niekoľkých sád booleovských premenných. Tieto booleovské premenné reprezentujú také skutočnosti, ako napr. či v i–tom kroku výpočtu sníma hlava j–te políčko, alebo či obsah j–teho políčka v i–tom kroku je a ($a \in \Gamma$) atď'. Navrhneme klauzuly, opisujúce prípustné výpočtové kroky stroja $\mathcal{M}$, ako aj jeho počiatočnú a akceptujúcu konfiguráciu. Priradenie hodnôt premenným, ktoré spĺňa všetky klauzuly, zodpovedá akceptujúcemu výpočtu $\mathcal{M}$ na w. Preto formula bude splniteľná práve vtedy, ak $\mathcal{M}$ akceptuje w.

Nech $\mathcal{M} = (Q, \Sigma, \Gamma, \triangleright, \sqcup, \delta, q_0)$. Vymenujeme najprv všetky používané booleovské premenné spolu s ich intuitívnou interpretáciou.

$\mathbf{Q_i^q}$ pre $0 \leq i \leq N,\ q \in Q$.
Premenná vyjadruje, či v i–tom kroku výpočtu je $\mathcal{M}$ v stave q.

$\mathbf{H_{ij}}$ pre $0 \leq i \leq N,\ 1 \leq j \leq N$.
Premenná vyjadruje, či v i–tom kroku výpočtu hlava sníma j–te políčko pásky.

$\mathbf{S_{ij}^a}$ pre $0 \leq i \leq N,\ 1 \leq j \leq N,\ a \in \Gamma$.
Premenná vyjadruje, či v i–tom kroku výpočtu j–te políčko pásky obsahuje symbol a.

Celkový počet premenných je $(N + 1)|Q| + N(N + 1) + N(N + 1)|\Gamma| = \mathcal{O}(p(|w|)^2)$. Hľadaná formula F má tvar

$$F = A \wedge B \wedge C \wedge D \wedge E \wedge G \wedge H$$

kde $A, \ldots, G$ sú booleovské formuly popisujúce $\mathcal{M}$ a jeho výpočet na w.

Spĺňajúce priradenie hodnôt premenným formuly F musí kódovať akceptujúci výpočet stroja $\mathcal{M}$ na w. Napríklad, práve jedna spomedzi premenných H_{11} až H_{1N} musí mať priradenú hodnotu true. Podobne pre premenné typu Q a S. Tieto podmienky sú vyjadrené formulami A, B a C. Formula A nadobúda hodnotu true práve ak poloha hlavy v každej konfigurácii je jednoznačne určená.

$$A = \bigwedge_{0 \leq i \leq N} \left((\bigvee_{1 \leq j \leq N} H_{ij}) \bigwedge_{1 \leq j < k \leq N} (\neg H_{ij} \vee \neg H_{ik}) \right)$$

Formula B nadobúda hodnotu true práve ak je jednoznačne určený stav každej konfigurácie.

$$B = \bigwedge_{0 \leq i \leq N} \left((\bigvee_{q \in Q} Q_i^q) \bigwedge_{\substack{p,q \in Q \\ p \neq q}} (\neg Q_i^p \vee \neg Q_i^q) \right)$$

Formula C nadobúda hodnotu true práve ak je jednoznačne určený obsah každého políčka v každej konfigurácii.

$$C = \bigwedge_{\substack{0\leq i\leq N \\ 1\leq j\leq N}} \left((\bigvee_{a\in\Gamma} S_{ij}^{a}) \bigwedge_{\substack{a,b\in\Gamma \\ a\neq b}} (\neg S_{ij}^{a} \vee \neg S_{ij}^{b}) \right)$$

V počiatočnej (nultej) konfigurácii je na páske zapísaný reťazec $\rhd w_1 \cdots w_n$ nasledovaný symbolmi $\sqcup$, hlava sníma symbol $\rhd$ a $\mathcal{M}$ je v počiatočnom stave q_0. Premenné zodpovedajúce počiatočnej konfigurácii preto musia nadobúdať relevantné booleovské hodnoty, čo overíme formulou D,

$$D = Q_0^{q_0} \wedge H_{01} \wedge S_{01}^{\rhd} \wedge \bigwedge_{1\leq j\leq |w|} S_{0(j+1)}^{w_j} \wedge \bigwedge_{|w|+2\leq j\leq N} S_{0j}^{\sqcup}.$$

Predpokladajme (bez straty všeobecnosti), že ak stroj $\mathcal{M}$ chce akceptovať vstup w, tak skôr než prejde do stavu q_{accept}, tak vymaže celý obsah pásky s výnimkou najľavejšieho symbolu $\rhd$ a nastaví hlavu na tento symbol. Potom akceptujúcu konfiguráciu možno zachytiť formulou

$$E = Q_N^{q_{accept}} \wedge H_{N1} \wedge S_{N1}^{\rhd} \wedge \bigwedge_{2\leq j\leq N} S_{Nj}^{\sqcup}.$$

Posledné dve formuly musia zachytiť najkomplikovanejšie podmienky. Je treba zaručiť, že spĺňajúce priradenie hodnôt premenným kóduje konfigurácie, ktoré tvoria výpočet, tj. že konfigurácie za sebou nasledujú v súlade s prechodovou reláciou. Formula G'_{ijpa} nadobúda hodnotu true práve ak i–tom kroku výpočtu sa stav, poloha hlavy a obsah j–teho políčka zmenia v súlade s prechodovou reláciou.

$$G = \bigwedge_{\substack{0\leq i<N \\ 1\leq j\leq N \\ p\in Q, a\in\Gamma}} G'_{ijpa}$$

$$G'_{ijpa} = \left(Q_i^{p} \wedge H_{ij} \wedge S_{ij}^{a}\right) \Longrightarrow \bigvee_{((p,a),(q,b,d))\in\delta} (Q_{i+1}^{q} \wedge H_{i+1,j+d} \wedge S_{i+1,j}^{b}) \,.$$

Formula H nadobúda hodnotu true práve ak obsah políčka, ktoré nebolo v i-tom kroku čítané, zostáva nezmenený.

$$H = \bigwedge_{\substack{0\leq i<N \\ 1\leq j\leq N \\ a\in\Gamma}} ((S_{ij}^{a} \wedge \neg H_{ij}) \Longrightarrow S_{i+1,j}^{a}) \,.$$

Formuly G a H nie sú v konjunktívnom normálnom tvare, ale použitím distributívneho zákona a DeMorganových pravidiel sa ľahko dajú do tohoto tvaru previesť. Dĺžka formuly F je $\mathcal{O}(p(n)^3)$ a dá sa skonštruovať z w v polynomiálnom čase. Naviac, každé spĺňajúce priradenie hodnôt zodpovedá akceptujúcemu výpočtu a naopak. □

Dôkaz NP–úplnosti redukciou

Vychádzajúc z NP–úplnosti problému splniteľnosti bola odvodená NP–úplnosť stoviek ďalších problémov. Ukážeme niektoré z nich. Ich spoločnou charakteristikou je, že

- pre každý prípad problému vyberáme fakt dosvedčujúci odpoveď „áno" z exponenciálne veľkej množiny do úvahy prichádzajúcich riešení a
- overiť, že vybrané riešenie skutočne dosvedčuje „áno" odpoveď je ľahké (t.j. vyžaduje si polynomiálny čas).

Efektívnosť nedeterministického výpočtu je založená na tom, že stačí (nedeterminsticky) vybrať jedno riešenie a overiť, či naozaj dosvedčuje odpoveď „áno". Z druhej strany, nevieme skonštruovať efektívnejší deterministický algoritmus než ten, ktorý je založený na postupnom prehľadávaní a preverovaní všetkých do úvahy prichádzajúcich riešení.

Skutočnosť, že nepoznáme efektívnu deterministickú simuláciu nedeterministického výpočtu spolu s predpokladom, že nájsť riešenie je podstatne ťažšie než overiť, že daný kandidát je správnym riešením, podporujú domnienku, že NP–úplné problémy sa nedajú riešiť deterministicky v polynomiálnom čase.

Prvý z problémov, o ktorom dokážeme, že je NP–úplný, je varianta problému splniteľnosti booleovskej formuly.

PROBLÉM kCNF
VSTUP: Booleovská formula F v konjunktívnej normálnej forme a taká, že každá klauzula obsahuje práve k literálov .
OTÁZKA: Je formula F splniteľná?

Lema 5.11 *Problém 3CNF je* NP*–úplný.*

Dôkaz Príslušnosť problému 3CNF do triedy NP sa opäť overí jednoducho. Ukážeme, ako možno problém CNF redukovať na problém 3CNF. Prípadom problému CNF je formula pozostávajúca z klauzulí; každú z nich musíme previesť na formulu obsahujúcu len klauzuly s tromi literálmi. Nech C_i je klauzula prípadu problému CNF. C_i transformujeme na formulu C_i' takto:

1. Ak $C_i = l_{i_1}$, tak $C_i' = l_{i_1} \vee l_{i_1} \vee l_{i_1}$.

2. Ak $C_i = l_{i_1} \vee l_{i_2}$, tak $C_i' = l_{i_1} \vee l_{i_2} \vee l_{i_1}$.

3. Ak $C_i = l_{i_1} \vee l_{i_2} \vee l_{i_3}$, tak $C_i' = l_{i_1} \vee l_{i_2} \vee l_{i_3}$.

4. Ak $C_i = l_{i_1} \vee l_{i_2} \vee \ldots \vee l_{i_k}$ pre $k > 3$, tak $C_i' = (l_{i_1} \vee l_{i_2} \vee y_{i_1}) \wedge (\neg y_{i_1} \vee l_{i_3} \vee y_{i_2}) \wedge \cdots (\neg y_{i_{k-4}} \vee l_{i_{k-2}} \vee y_{i_{k-3}}) \wedge (\neg y_{i_{k-3}} \vee l_{i_{k-1}} \vee l_{i_k})$, kde y sú nové premenné.

Ľahko sa overí, že popísanú transformáciu možno vypočítať v polynomiálnom čase.

Ukážeme, že ak t je priradenie hodnôt premenným, ktoré spĺňa klauzulu C_i, tak z neho môžeme skonštruovať priradenie t' spĺňajúce formulu C_i'. Problém je triviálny ak C_i obsahuje maximálne tri literály. Preto nech $C_i = l_{i_1} \vee l_{i_2} \vee \ldots \vee l_{i_k}$, kde $k > 3$. Keďže t spĺňa C_i, tak existuje literál l_{i_j}, ktorý má priradenú hodnotu true. Ostatným premenným formuly C_i' priradíme hodnoty v závislosti od indexu j:

1. $j = 1, 2$: l_{i_j} sa vyskytuje v klauzule tvaru $(l_{i_1} \vee l_{i_2} \vee y_{i_1})$; v tomto prípade všetkým y–premenným priradíme hodnotu false.

2. $j = k - 1, k$: l_{i_j} sa vyskytuje v klauzule tvaru $(\neg y_{i_{k-3}} \vee l_{i_{k-1}} \vee l_{i_k})$; v tomto prípade všetkým y–premenným priradíme hodnotu true.

3. $2 < j < k - 1$: l_{i_j} sa vyskytuje v klauzule tvaru $(\neg y_{i_{j-2}} \vee l_{i_j} \vee y_{i_{j-1}})$; v tomto prípade prvým $j - 2$ y–premenným priradíme hodnotu true, zatiaľ čo ostatným priradíme hodnotu false.

Naopak, ak máme priradenie spĺňajúce formulu C_i' , tak to isté priradenie, obmedzené len na pôvodné premenné, spĺňa formulu C_i. Preto problém 3CNF je NP–úplný. □

Problém 3CNF je zaujímavý hneď z dvoch dôvodov. Vďaka svojej jednoduchosti sa často využíva ako referenčný problém pri dôkaze NP–úplnosti redukciou. Zároveň predstavuje „hraničný“ problém. Ak by sme totižto problém CNF zúžili na formuly, kde každá klauzula obsahuje iba *dva* literály, tak problém prestane byť NP–úplným.

Lema 5.12 *Problém 2CNF patrí do triedy* P.

Dôkaz Nech F je formula v 2–konjunktívnom normálnom tvare. Bez straty obecnosti môžeme predpokladať, že každá jej klauzula má práve dva literály (v opačnom prípade nahradíme klauzulu l klauzulou $l \vee l$).

Na začiatku deklarujeme všetky klauzuly ako nesplnené. Vyberieme ľubovoľnú premennú x a priradíme jej hodnotu true . Následne sa snažíme priradiť hodnoty ostatným premenným a to na základe nasledujúceho pravidla:

Vezmi ľubovoľnú nesplnenú klauzulu $(l_l \vee l_r)$ takú, že jeden z literálov, povedzme l_l, už má priradenú hodnotu. Ak l_l má priradenú hodnotu true, tak klauzulu deklaruj ako splnenú. Ak l_l má priradenú hodnotu false, tak literálu l_r priraď hodnotu true a deklaruj klauzulu ako splnenú.

Počas výpočtu môže nastať niektorá z nasledujúcich troch situácií.

1. Konflikt vznikne v okamihu, keď sa snažíme priradiť hodnotu true literálu, ktorý už má priradenú hodnotu false. To ale znamená, že počiatočné priradenie pravdivostnej hodnoty premennej x nebolo správne. Preto priradíme premennej x hodnotu false a celý výpočet opakujeme odznovu. Ak aj potom vznikne konflikt, tak formula F je nesplniteľná.

2. Nevznikne žiaden konflikt a všetky premenné majú priradenú hodnotu. Potom formula F je splniteľná.

3. Nevznikne žiaden konflikt, ale nie všetky premenné majú priradené hodnoty. Takáto situácia môže nastať iba vtedy, ak literály v nesplnených formulách nemajú priradenú žiadnu hodntou. V takomto prípade môžeme ignorovať všetky už splnené klauzuly a pracovať s redukovanou formulou. Zrejme redukovaná formula je splniteľná práve ak pôvodná formula je splniteľná. Splniteľnosť redukovanej formuly overíme rekurzívne.

Ľahko overíme, že načrtnutý algoritmus pracuje v čase $\mathcal{O}(mn)$, kde n jepočet premenných formuly F a m je počet jej klauzulí. □

Nasledujúce NP–úplné problémy sa týkajú testovania vlastností grafov. Prvý z nich je problém vrcholového pokrytia grafu.

PROBLÉM k–VRCHOLOVÉ POKRYTIE
VSTUP: Neorientovaný graf $G = (V, H)$ a konštanta $k \in \mathbb{N}$.
OTÁZKA: Existuje pokrytie grafu G nanajvýš k vrcholmi? Inak povedané, zisťujeme, či existuje k–prvková množina $M \subseteq V$ taká, že pre každú hranu $(i, j) \in H$ aspoň jeden z vrcholov i, j patrí do M.

Lema 5.13 *Problém k–VRCHOLOVÉ POKRYTIE je* NP*–úplný.*

Dôkaz Problém k–VRCHOLOVÉ POKRYTIE zrejme patrí do NP. Nedeterministický TS, ktorý dostane na vstup kód grafu G a číslo k, (nedeterministicky) vyberie k jeho vrcholov a (deterministicky) overí, či pokrývajú každú hranu grafu.

Navrhneme redukciu problému splniteľnosti CNF na problém pokrytia. Redukcia priradí booleovskej formule F v konjunktívnej normálnej forme neorientovaný graf G a číslo k také, že F je splniteľná práve vtedy ak graf G je možné pokryť k vrcholmi.

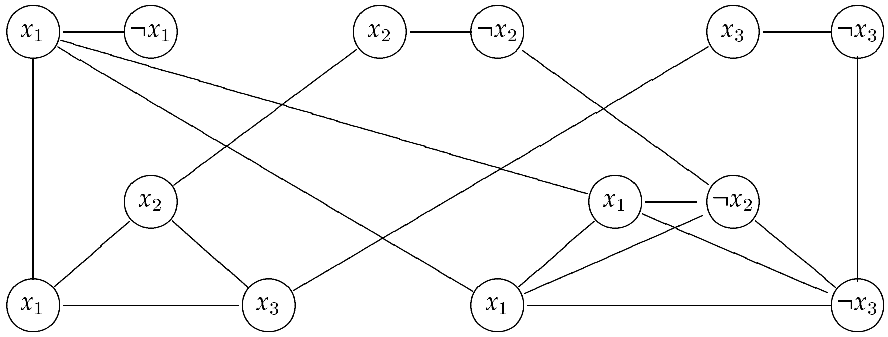
Nech $x_1, x_2, \dots, x_n$ sú premenné formuly F, nech m je počet jej klauzulí a n_i $(1 \leq i \leq m)$ je počet literálov i–tej klauzuly. Definujme graf $G = (V, H)$ takto:

1. Pre každú premennú x_i formuly F obsahuje graf G dvojicu vrcholov x_i^t a x_i^f a hranu (x_i^t, x_i^f). Na pokrytie tejto je hrany je potrebný (a postačujúci) jeden vrchol. Intuitívne; vybraný vrchol bude zodpovedať priradeniu hodnoty premennej x_i.

2. Pre každú klauzulu C_j formuly F obsahuje G kompletný podgraf G_j s n_j vrcholmi, z ktorých každý zodpovedá jednému literálu klauzuly. Na pokrytie hrán tejto kliky je potrebných (a postačujúcich) $n_j - 1$ vrcholov. Intuitívne; zvyšný vrchol kliky zodpovedá tomu literálu klauzuly, ktorý bude mať priradenú pravdivostnú hodnotu true.

3. Ak x_i resp. $\neg x_i$ je literál formuly F, tak G obsahuje hranu spájajúcu vrchol zodpovedajúci literálu x_i resp. $\neg x_i$ s vrcholom x_i^t resp. x_i^f.

Napríklad, formule $(x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee \neg x_2 \vee \neg x_3 \vee x_1)$ priradíme graf



Číslo k definujeme ako

$$k \stackrel{\text{def}}{=} n + (n_1 - 1) + (n_2 - 1) + \ldots + (n_m - 1).$$

Ľahko overíme, že uvedenú transformáciu možno vypočítať v polynomiálnom čase a že F je splniteľná práve ak G je možné pokryť k vrcholmi. Skutočne: ak F je splniteľná, tak graf možno pokryť takto:

1. Pre každú premennú x_i, vyberieme vrchol x_i^t ak premennej x_i bola priradená hodnota true; inak vyberieme vrchol x_i^f.

2. Pre každú kaluzulu C_j vyberieme do pokrytia všetkých n_j vrcholov grafu G_j s výnimkou jedného, ktorý je spojený s vrcholom, ktorý sme už zaradili do pokrytia. Existencia takéhohoto vrcholu plynie z toho, že pracujeme so spĺňajúcim priradením hodnôt premenným.

Naopak, ak máme pokrytie grafu G k vrcholmi, tak spĺňajúce priradenie hodnôt premenným formuly F je takéto: premennej x_i priradíme hodnotu true práve ak x_i^t patrí do pokrytia, inak jej priradíme hodnotu false. □

Ďalším NP–úplným problémom je k–KLIKA, pretože CNF $\leq$ k–KLIKA (lema 5.3) a k–KLIKA patrí do triedy NP. Alternatívny spôsob dôkazu NP–úplnosti problému k–KLIKA poskytuje redukcia problému k–VRCHOLOVÉ POKRYTIE na tento problém. Pre daný graf $G = (V, H)$ definujeme k nemu komplementárny graf ako $G^c = (V, H^c)$, kde $H^c = \{(i,j) \mid i,j \in V;\ i \neq j;\ (i,j) \notin H\}$. Zrejme pre ľubovoľné $V' \subseteq V$ je V' vrcholovým pokrytím pre G vtedy a len vtedy ak vrcholy z $V \setminus V'$ tvoria kompletný podgraf grafu G^c. Hľadaná redukcia spočíva v transformovaí grafu G a čísla k na komplementárny graf G^c a číslo $|V| - k$.

S problémom vrcholového pokrytia úzko súvisí problém nezávislej množiny.

PROBLÉM k–NEZÁVISLÁ MNOŽINA
VSTUP: Neorientovaný graf $G = (V, H)$ a konštanta $k \in \mathbb{N}$.
OTÁZKA: Existuje k prvková množina $M \subseteq V$ taká, že pre každé dva prvky $i, j \in M$ platí $(i, j) \notin H$?

Redukcia problému k–KLIKA na problém k–NEZÁVISLÁ MNOŽINA priradí dvojici graf G a číslo k komplementárny graf G^c a to isté číslo k. Pretože problém k–NEZÁVISLÁ MNOŽINA patrí do triedy NP, jedná sa o NP–úplný problém.

Nasledujúci np–úplný problém sa týka farbenia grafov.

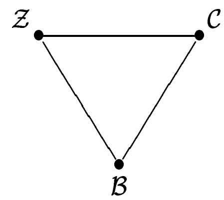
PROBLÉM k–ZAFARBENIE GRAFU
VSTUP: Neorientovaný graf $G = (V, H)$ a konštanta $k \in \mathbb{N}$.
OTÁZKA: Existuje také priradenie farieb $1, 2, \ldots, k$ vrchlom grafu, že každé dva susedné vrcholy majú rôzne farby?

Podobne ako pre problém kCNF je aj pre problém k–ZAFARBENIE GRAFU hraničnou hodnotou $k = 3$. Pre $k = 2$ je problém jednoduchý: graf je zafarbiteľný dvoma farbami práve vtedy, keď je bipartitný, čo je práve vtedy, ak nemá cyklus nepárnej dĺžky. Tieto vlastnosti sa dajú overiť v polynomiálnom čase prehľadávaním do hĺbky resp. do šírky.

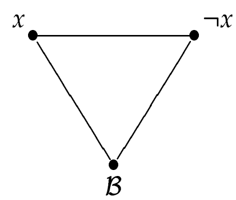
Lema 5.14 *Problém k–ZAFARBENIE GRAFU je pre $k \geq 3$ NP–úplný.*

Dôkaz Ukážeme, že pre $k = 3$ je problém NP–úplný. Jeho príslušnosť do triedy NP sa overí jednoducho. Navrhneme redukciu CNF $\leq$ 3–ZAFARBENIE GRAFU. Nech F je booleovská formula v konjunktívnom normálnom tvare a nech $x_1, \ldots, x_n$ sú jej premenné. Skonštruujeme graf G, ktorý bude zafarbiteľný 3 farbami práve ak F je splniteľná.

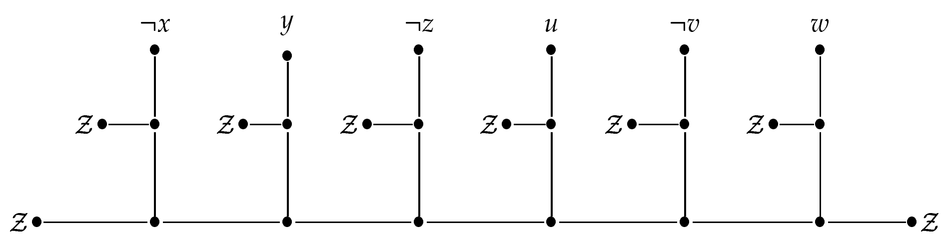
Graf G bude mať tri špeciálne vrcholy $\mathcal{Z}, \mathcal{C}, \mathcal{B}$, spojené do trojuholníka (kompletný graf o troch vrcholoch). V 3–zafarbení grafu musia mať tieto tri vrcholy rôzne farby, povedzme zelenú, červenú a bielu.



Pre každý premennú x formuly F bude graf obsahovať vrcholy x a $\neg x$ a hrany tvoriace kompletný graf s vrcholmi $x, \neg x, \mathcal{B}$. V 3–zafarbení grafu vrcholy zodpovedajúce literálom x a $\neg x$ musia mať zelenú alebo červenú farbu a obidva nesmú mať tú istú farbu. Intuitívne; správne 3–zafarbenie grafu reprezentuje spĺňajúce priradenie hodnôt premenným formuly, pričom zelené literály sú pravdivé a červené nepravdivé.



Aby sme dokončili konštrukciu grafu potrebujeme k nemu pridať pre každú klauzulu formuly podgraf. Tvar podgrafu pre klauzulu $(x \vee y \vee \neg z \vee u \vee \neg v \vee w)$ je naznačený na obrázku. Vrcholy označené na obrázku ako $\mathcal{Z}$ predstavujú v skutočnosti ten istý vrchol grafu, konkrétne vrchol $\mathcal{Z}$.



Tento podgraf má tú vlastnosť, že zafarbenie vrcholov v hornom riadku zelenou alebo červenou farbou sa dá rozšíriť na 3–zafarbenie celého podgrafu vtedy a len vtedy, ak aspoň jeden vrchol v hornom riadku je zelený. Ak všetky vrcholy v hornom riadku sú zafarbené na červeno, potom všetky vrcholy v strednom riadku susediace s vrcholmi horného riadku musia byť zafarbené na bielo. Štartujúc zľava, vrcholy v spodnom riadku musia byť zafarbené striedavo červenou a zelenou farbou. To vedie ku konfliktu pri farbení posledného vrcholu v spodnom rade (ak počet literálov v klauzule je nepárny, tak najpravejší vrchol v spodnej rade označíme namiesto $\mathcal{Z}$ ako $\mathcal{C}$).

Naopak, predpokladajme, že niektorý z vrcholov hornom rade bol zafarbený zelenou farbou. Vyberme si tento zelený vrchol. Zafarbime vrchol v strednom rade s ním susediaci červenou farbou a s ním susediaci vrchol v spodnom rade bielou farbou. Všetky ostané vrcholy v strednom rade zafarbíme na bielo. Štartujúc z oboch koncov farbíme vrcholy spodného radu červenou a zelenou farbou tak, ako nám to graf dovolí.

Preto v 3–zafarbení grafu G musí mať každý podraf zodpovedajúci klauzule aspoň jeden zelený literál a priradenie hodnoty truezeleným literálom určuje spĺňajúce priradenie hodnôt premenným formuly F. Naopak, ak F je splniteľná, zafarbenie vrcholov zodpovedajúcim pravdivým literálom na zeleno a ostatných na červeno indikuje 3–zafarbenie grafu.

Problém 3–ZAFARBENIE GRAFU ľahko rozšírime na problém k–ZAFARBENIE GRAFU tak, že do grafu pridáme kompletný graf s $k-3$ vrcholmi a hrany spájajúce vrcholy tejto kliky s každým iným vrcholom grafu. □

S mnohými NP–úplnými úlohami sa môžeme stretnúť pri testovaní vlastností množín. Uvedieme niekoľko z nich.

PROBLÉM PRESNÉ POKRYTIE
VSTUP: Konečná množina X a systém S podmnožín množiny X.
OTÁZKA: Existuje podmnožina $S' \subseteq S$ taká, že každý element X je prvkom práve jednej množiny systému S'?

Lema 5.15 *Problém PRESNÉ POKRYTIE je* NP*–úplný.*

Dôkaz Príslušnosť problému do triedy NPsa overí jednoducho. Navrhneme redukciu 3–ZAFARBENIE GRAFU $\leq$ PRESNÉ POKRYTIE.

Majme daný neorientovaný graf $G = (V, H)$. Pre každý vrchol $u \in V$ označme $N(u)$ množinu vrcholov susediacich s u. Z neorientovanosti grafu G máme $u \in N(v)$ práve ak $v \in N(u)$. Budeme uvažovať tri farby: modrú, zelenú a bielu. Našim cieľom je ku grafu G skonštruovať prípad (X, S) problému pokrytia.

Najprv definujeme množinu X. Do množiny X zaradíme každý vrchol $u \in V$ spolu s ďalšími $3(|N(u)| + 1)$ prvkami. Týchto $3(|N(u)| + 1)$ prvkov rozdelíme do troch disjunktných množín, z ktorých každá bude mať $|N(u)| + 1$ prvkov. Každá množina zodpovedá jednej farbe; označme ich $S_u^{modrá}, S_u^{zelená}, S_u^{biela}$. Pre každú farbu c vyberieme jeden špeciálny prvok p_u^c z S_u^c. Ostatných $|N(u)|$ prvkov množiny S_u^c asociujeme jedno–jednoznačným spôsobom s prvkami množiny $N(u)$. Nech q_{uv}^c označuje ten prvok množiny S_u^c, ktorý je asociovaný s prvkom $v \in N(u)$.

Definujeme systém S. S obsahuje všetky dvojprvkové množiny

$$\{u, p_u^c\} \tag{5.1}$$

a všetky množiny

$$S_u^c \tag{5.2}$$

pre $u \in V$ a farbu $c \in \{$modrá, zelená, biela$\}$. Okrem toho S obsahuje dvojprvkové množiny tvaru

$$\{q_{uv}^c, q_{uv}^{c'}\} \tag{5.3}$$

pre všetky $(u, v) \in H$ a $c, c' \in \{$modrá, zelená, biela$\}$, $c \neq c'$.

Ukážeme, že množina X má presné pokrytie práve vtedy, ak existuje zobrazenie $\Lambda : V \to \{$modrá, zelená, biela$\}$, ktoré je 3–zafarbením grafu G.

Predpokladajme, že Λ je 3–zafarbenie grafu G. Potom množinu X môžeme pokryť systémom množín S'. Pre každý vrchol $u \in V$ obsahuje S' množiny $\{u, p_u^{\Lambda(u)}\}$ a S_u^c pre $c \neq \Lambda(u)$. Tieto množiny pokryjú všetky prvky X okrem tých, ktoré sú tvaru $q_{uv}^{\Lambda(u)}$, kde $(u, v) \in H$. Pre každú hranu (u, v) preto pridáme do S' množinu $\{q_{uv}^{\Lambda(u)}, q_{uv}^{\Lambda(v)}\}$, ktorá určite patrí do S, lebo $\Lambda(u) \neq \Lambda(v)$. Tým sme pokryli každý prvok X práve jednou množinou zo systému S'.

Naopak, predpokladajme, že S' je presné pokrytie množiny X. Každý prvok u je pokrytý práve jednou množinou z S'; tá musí byť tvaru $\{u, p_u^c\}$ pre nejaké c. Položme $\Lambda(u) \stackrel{\text{def}}{=} c$. Tvrdíme, že takto definované zobrazenie je 3–zafarbením grafu G. Musíme ukázať, že ak $(u, v) \in H$, tak $\Lambda(u) \neq \Lambda(v)$. Pretože $\{u, p_u^{\Lambda(u)}\} \in S'$, tak žiadne p_u^c pre $c \neq \Lambda(u)$ nemôže byť pokryté množinou tvaru 5.1 (prvok u by bol pokrytý dva krát). Pokrytie teda obsahuje množiny tvaru 5.2, pretože tie ako jediné zo zvyšných množín obsahujú prvky p_u^c. Množiny $\{u, p_u^{\Lambda(u)}\}$ a S_u^c pre $c \neq \Lambda(u)$ pokrývajú všetky prvky s výnimkou tých, ktoré majú tvar $q_{uv}^{\Lambda(u)}$, kde $(u, v) \in H$. Jediný spôsob, ako ich pokryť, je vziať množiny tvaru 5.3. Z konštrukcie S, tieto sú tvaru $\{q_{uv}^{\Lambda(u)}, q_{uv}^{\Lambda(v)}\}$ pre $(u, v) \in H$ a $\Lambda(u) \neq \Lambda(v)$. □

Nasledujúce tri problémy—problém batohu, problém sumy a problém rozdelenia—spolu veľmi úzko súvisia.

PROBLÉM H–BATOH
VSTUP: Konečná množina S položiek (objektov), váhová funkcia $v : S \to \mathbb{N}$, hodnotová funkcia $h : S \to \mathbb{N}$, váhový limit $V \in \mathbb{N}$ a očakávaný zisk $H \in \mathbb{N}$.
OTÁZKA: Existuje taká podmnožina $S' \subseteq S$, že

$$\sum_{a \in S'} v(a) \leq V$$

$$\sum_{a \in S'} h(a) \geq H \ ?$$

Názov úlohy je odvodený zo situácie, keď do batohu fixovanej váhovej kapacity chceme uložiť položky rôznej váhy tak, aby hodnota uložených vecí bola aspoň H.

PROBLÉM SUBSET SUM
VSTUP: Konečná množina S položiek, váhová funkcia $v : S \to \mathbb{N}$ a cieľová hodnota $V \in \mathbb{N}$.
OTÁZKA: Existuje taká podmnožina $S' \subseteq S$, že

$$\sum_{a \in S'} v(a) = V \ ?$$

Alternatívna formulácia problému je: Predpokladajme, že množina S má n prvkov; označme ich váhy porade $v_1, \ldots, v_n$. Existuje booleovský vektor $(x_1, \ldots, x_n)$ taký, že

$$\sum_{i=1}^{n} x_i v_i = V \ ?$$

Jedná sa vlastne o zjednodušenú verziu problému H–BATOH.

PROBLÉM PARTITION
VSTUP: Konečná množina S položiek a váhová funkcia $v : S \to \mathbb{N}$.
OTÁZKA: Existuje taká podmnožina $S' \subseteq S$, že

$$\sum_{a \in S'} v(a) = \sum_{a \in S - S'} v(a) \ ?$$

Redukciu PARTITION $\leq$ SUBSET SUM nájdeme jednoducho: stačí vziať

$$V = \frac{1}{2} \sum_{a \in S} v(a).$$

Analogicky redukciu SUBSET SUM $\leq$ PARTITION získame pridaním dvoch nových položiek váhy $N - V$ a $N - (\Sigma - V)$, kde

$$\Sigma = \sum_{a \in S} v(a)$$

a N je dostatočne veľké číslo (stačí $N > \Sigma$). Číslo N je vybrané tak, aby nové dve položky nemohli patriť do tej istej podmnožiny, pretože prevážia všetky ostatné položky. Teraz sa pýtame, či novovytvorená množina položiek sa dá rozdeliť na dve množiny rovnakej váhy (a to práve N). Ak z rozdelenia vynecháme dve nové položky, dostaneme rozdelenie pôvodnej množiny položiek na podmnožiny váhy V a $\Sigma - V$.

Oba problémy SUBSET SUM a PARTITION sú redukovateľné na problém batohu. Redukciu PARTITION $\leq$ H–BATOH dostaneme voľbou $v = h$ a $V = H = \frac{1}{2}\Sigma$.

Nakoniec ukážeme, že všetky tri problémy sú NP–úplné a to tak, že navrhneme redukciu PRESNÉ POKRYTIE $\leq$ SUBSET SUM.

Lema 5.16 *Problémy H–BATOH, SUBSET SUM a PARTITION sú* NP*–úplné.*

Dôkaz Príslušnosť problémov do triedy NP sa overí jednoducho. NP–ťažkosť problémov preukážeme nájdením redukcie 3CNF $\leq$ SUBSET–SUM.

Nech F je formula v 3–konjunktívnej normálnej forme, $F = F_1 \wedge \cdots \wedge F_k$, kde každé F_i je klauzula s 3 literálmi z množiny $\{x_1, \ldots, x_m, \neg x_1, \ldots, \neg x_m\}$. Množina S bude obsahovať $2m + 2k$ prvkov, ktorých váhy označíme $v_1, \ldots, v_{2m+2k}$. Hodnota V je prirodzené číslo, ktorého zápis v desiatkovej sústave má $(m + k)$ číslic, všetky rovné 3

$$V = \underbrace{3 \ldots 3}_{m+k} .$$

Váhy $v_1, \ldots, v_{2m+2k}$ sú $(m + k)$–ciferné čísla, ktoré poporadí označíme ako

$$\overline{x_1}, \ldots, \overline{x_m}, \quad \neg\overline{x_1}, \ldots, \neg\overline{x_m}, \quad y_1, \ldots, y_k, \quad y'_1, \ldots, y'_k$$

a definujeme nasledovne (notácia použitá v predchádzajúcom riadku naznačuje súvislosť medzi týmito číslami na jednej strane a literálmi $x_1, \ldots, x_m, \neg x_1, \ldots, \neg x_m$ a klauzulami $F_1, \ldots, F_k$ na strane druhej):

1. Pre $i = 1, \ldots, m$, i–ta cifra zprava v $\overline{x_i}$ ako aj v $\neg\overline{x_i}$ je 3. Pre $j = 1, \ldots, k$, $(m + j)$–ta cifra zprava v $\overline{x_i}$ (v $\neg\overline{x_i}$) je 1 práve ak F_j obsahuje x_i (ak F_j obsahuje $\neg x_i$). Všetky ostatné číslice v $\overline{x_i}$ a v $\neg\overline{x_i}$ sú 0.

   Upozorňujeme, že $\overline{x_i}$ obsahuje, v desiatkovom zápise, úplnú a ľahko dekódovateľnú informáciu o tom, ktorej premennej zodpovedá (konkrétne x_i) a v ktorých klauzulách sa x_i vyskytuje. To isté platí pre $\neg\overline{x_i}$. Preto čísla $\overline{x_1}, \ldots, \overline{x_m}, \neg\overline{x_1}, \ldots, \neg\overline{x_m}$ obsahujú kompletnú informáciu o formule F.

2. Pre $j = 1, \ldots, k$, $(m+j)$–ta cifra zprava v y_j je 1. Všetky ostatné číslice sú 0. Naviac, $y_i = y'_i$ pre všetky $0 \leq i \leq k$.

K formule $F = (x_1 \vee x_2 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_2 \vee x_3)$ by sme takto skonštruovali $(k = 2,\ m = 3)$

$$\begin{array}{llll} c = 3333 & \overline{x_1} = 01003 & \neg\overline{x_1} = 10003 & y_1 = 01000 = y'_1 \\ & \overline{x_2} = 01030 & \neg\overline{x_2} = 10030 & y_2 = 10000 = y'_2 \\ & \overline{x_3} = 10300 & \neg\overline{x_3} = 01300 & \end{array}$$

Zrejme $V = \overline{x_1} + \overline{x_2} + \overline{x_3} + y_1 + y'_1 + y_2$.

Zostáva nám ukázať, že problém SUBSET SUM S s váhami $v_1, \ldots, v_{2m+2k}$ a hodnotou V má riešenie vtedy a len vtedy, ak formula F je splniteľná.

1. Nech F je splniteľná. Fixujme spĺňajúce priradie α pre F. Pre každé $i = 1, \ldots, m$ zaradíme do riešenia S prvok s váhou $\overline{x_i}$ práve ak literál x_i $(\overline{x_i})$ má hodnotu true (false) v priradení α. Suma všetkých týchto váh má posledných m číslic rovných 3; každá z prvých k číslic je buď 1, 2 alebo 3. Pripočítaním vhodného počtu y a y' dostaneme presne V.

2. Nech problém S má riešenie. To znamená, že existuje taká podmnožina $\overline{x_1}, \ldots, y'_k$, ktorá je riešením S. To je možné len vtedy, ak pre každé $i = 1, \ldots, m$ práve jedno z čísiel $\overline{x_i}$ alebo $\neg\overline{x_i}$ je v sume rovnej V. To číslo určuje spĺňajúce priradenie formuly F.

□

Pre ilustráciu uvádzame ešte niekoľko ďaľších NP–úplných problémov.

PROBLÉM H–BIN PACKING
VSTUP: Konečná množina S položiek, váhová funkcia $v : S \to \mathbb{N}$ a k košov váhovej kapacity $V \in \mathbb{N}$.
OTÁZKA: Je možné všetky položky množiny S uložiť do k košov tak, aby nebola prekročená ich váhová kapacita?

PROBLÉM CELOČÍSELNÉ PROGRAMOVANIE
VSTUP: $k \times l$ matica A nad $\mathbb{Z}$, $b \in \mathbb{Z}^k$.

OTÁZKA: Existuje $y \in \mathbb{Z}^l$ také, že $Ay \geq b$?

Ľahko sa zostrojí redukcia SUBSET SUM $\leq$ CELOČÍSELNÉ PROGRAMOVANIE. Pre množinu S, váhovú funkciu $v : S \to \mathbb{N}$ a hodnotu $V \in \mathbb{N}$ existuje pozitívne riešenie práve ak prípad

$$0 \leq x_a \leq 1,\ a \in S$$

$$\sum_{a \in S} v(a) x_a = V$$

celočíselného programovania má celočíslené riešenie.
Ak pripustíme, aby hľadaný vektor y obsahoval racionálne čísla, tak problém patrí do triedy P.

NP–úplnými sú aj problémy Hamiltonovskej cesty a rozhodovacia verzia problému obchodného cestujúceho.

PSEUDOPOLYNOMIÁLNE ALGORITMY A NP–ÚPLNOSŤ

Uvažujme ešte raz problém batohu.

Lema 5.17 *Existuje algoritmus, ktorý riešenie každého prípadu problému H–BATOH nájde v čase $\mathcal{O}(nV)$, kde n je počet položiek prípadu a V váhový limit.*

Dôkaz Nech $S = \{a_1, \dots, a_n\}$. Definujme $W(v, i)$ ako najväčšiu hodnotu, ktorú môžeme dosiahnuť výberom spomedzi prvých i položiek tak, že ich váha je práve v. Ľahko vidno, že nV položiek tabuľky $W(v, i)$ možno postupne vypočítať, začínajúc od najmenšieho i, každú položku v konštantnom čase:

$$W(v, i+1) = \max\{W(v, i), h(a_{i+1}) + W(v - v(a_{i+1}), i)\}.$$

Položky $W(v, 0) = 0$ pre všetky v. Daný prípad problému batohu má riešenie „áno", ak v konštruovanej tabuľke existuje položka väčšia alebo rovná H. Preto stačí, aby sme si počas výpočtu pamätali najväčšiu dosiaľ vypočítanú hodnotu a nakoniec ju porovnali s H. $\square$

Uvedený algoritmus však (aj keď sa to naprvý pohľad môže zdať) nie je polynomiálny. Jeho časová zložitosť nV nie je funkciou polynomiálnou od dĺžky vstupu. Dĺžka vstupu je totižto rádovo $n \log V$. Jedná sa o tzv. pseudopolynomiálny algoritmus. Pseudopolynomiálnym nazveme každý algoritmus, ktorého časová zložitosť by bola polynomiálna v prípade, že všetky vstupné údaje sú zapísané v unárnej notácii. Pseudopolynomiálne algoritmy sú zdrojom častých nedorozumení.

Ak problém zostáva NP–úplný aj v prípade, že požadujeme, aby všetky prípady problému rozsahu n obsahovali čísla veľkosti nanajvýš $p(n)$, kde p je polynóm, tak hovoríme o *silnej* NP–*úplnosti*. Všetky problémy, pre ktoré sme doteraz ukazovali NP–úplnosť sú silne NP–úplné— práve s výnimkou problému H–BATOH. Preto nie je prekvapujúce, že spomedzi všetkých týchto problémov jedine H–BATOH môže byť riešený pseudopolynomiálnym algoritmom. Za predpokladu P $\neq$ NP sa silne NP–úplné problémy nedajú riešiť žiadnym pseudopolynomiálnym algoritmom.

Kapitola 6

P **versus** NP

6.1 Alternatívna charakterizácia problémov z NP

Triedu NP sme definovali pomocou nedeterminisitických Turingovych strojov. Ukážeme, ako inak môžeme definovať túto zložitostnú triedu.

Nech Σ je abeceda a nech $R \subseteq \Sigma^* \times \Sigma^*$ je binárna relácia. Reláciu R nazveme *polynomiálne rozhodnuteľnou*, ak existuje Turingov stroj rozhodujúci jazyk $\left\{ \langle x, y \rangle \;\middle|\; (x, y) \in R \right\}$ v polynomiálnom čase. Reláciu R nazveme *polynomiálne vyváženou* ak existuje konštanta $k \in \mathbb{N}$ taká, že $(x, y) \in R$ implikuje $|y| \leq |x|^k$. Inak povedané, dĺžka druhej komponenty je vždy ohraničená polynómom dĺžky prvej komponenty.

Lema 6.1 *Nech $L \subseteq \Sigma^*$ je jazyk. $L \in \mathsf{NP}$ vtedy a len vtedy, ak existuje polynomiálne rozhodnuteľná a polynomiálne vyvážená relácia R taká, že*

$$L = \left\{ x \mid (x, y) \in R \textit{ pre nejaké } y \right\}.$$

Turingov stroj $\mathcal{M}$ rozhodujúci reláciu R sa nazýva overovateľ pre *L.*

Dôkaz Predpokladajme, že existuje relácia R uvedených vlastností. Potom jazyk L môže byť rozhodovaný nasledujúcim nedeterministickým, polynomiálne časovo ohraničeným strojom $\mathcal{T}$. Pre vstup x uhádne $\mathcal{T}$ reťazec y dĺžky nanajvýš $|x|^k$. Potom, využijúc stroj $\mathcal{M}$, rozhodujúci reláciu R, overí, či $(x, y) \in R$. Ak áno, akceptuje, inak zamietne. Zrejme akceptujúci výpočet na vstupe x existuje práve vtedy, ak $x \in L$.

Z druhej strany; nech $L \in \mathsf{NP}$, tzn. existuje nedeterministický Turingov stroj $\mathcal{N}$ rozhodujúci L v čase $|x|^k$ pre nejaké k. Definujeme reláciu R takto: $(x, y) \in R$ práve vtedy, ak y je kód akceptujúceho výpočtu stroja $\mathcal{N}$ na vstupe x. Polynomiálna vyváženosť relácie R plynie z polynomiálnej časovej zložitosti stroja $\mathcal{N}$. Zároveň je R polynomiálne rozhodnuteľná, lebo v polynomiálnom čase vieme overiť, či y je skutočne kódom akceptujúceho výpočtu stroja $\mathcal{N}$ na x. Naviac, ak $\mathcal{N}$ rozhoduje L, tak $L = \left\{ x \mid (x, y) \in R \text{ pre nejaké } y \right\}$. □

Lema 6.1 poskytuje nový pohľad na triedu NP. Ak jazyk L patrí do NP, tak každé slovo $x \in L$) má krátky (polynomiálne veľký) *certifikát* (dôkaz príslušnosti) y d dosvedčujúci, že $x \in L$. Naviac, korektnosť vybraného ceritifikátu sa ľahko (v polynomiálnom čase) overí. Napríklad pre problém splniteľnosti booleovskej formuly je certifikátom spĺňajúce priradenie hodnôt premenným. V probléme Hamiltonovského cyklu je certifikátom hamiltonovský cyklus. Zdôrazňujeme, že nová charakterizácia triedy NP vôbec nepoužíva pojem nedeterminizmu.

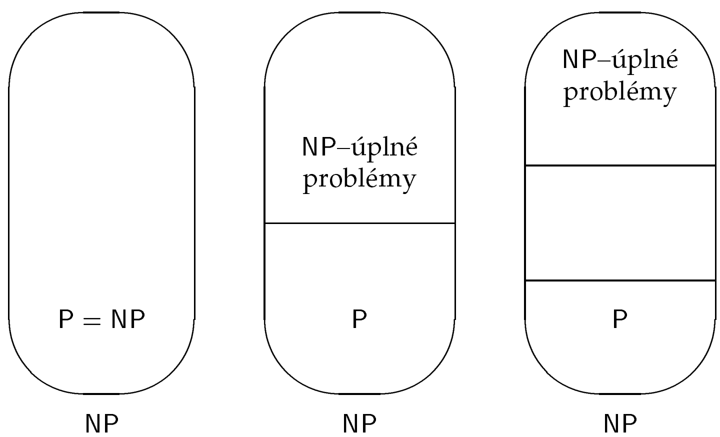
Trieda NP je z praktického hľadiska nepochybne veľmi dôležitá—obsahuje prevažnú väčšinu problémov, ktoré potrebujeme prakticky riešiť. Lema 6.1 sčasti poskytuje vysvetlenie, prečo tomu tak je: mnoho výpočtových problémov totižto spočíva v úlohe navrhnúť objekt s určitými vlastnosťami. Tieto matematické objekty sú abstrakciou reálnych objektov a preto je prirodzené, že nemôžu mať enormnú veľkosť. Pritom tieto objekty sú vlastne certifikátmi, o ktorých hovorí lema 6.1.

Z druhej strany, trieda P sa dá charakterizovať ako trieda problémov, kde certifikát (dôkaz príslušnosti) prípadu sa dá ľahko (v polynomiálnom čase) *nájsť*. Na P $\stackrel{?}{=}$ NPproblém sa preto môžeme pozerať ako na otázku čo je ťažšie: nájsť dôkaz alebo overiť dôkaz. Intuitívne sa zdá odpoveď na otázku jasná—ale len do chvíle, než sa ju pokúsime dokázať.

6.2 Vzťah tried P a NP

Význam pojmu NP–úplnosti pri klasifikácii problémov je nepopierateľný; v súčastnosti sú známe stovky NP–úplných problémov. Avšak aj napriek veľkému úsiliu existujú problémy, na ktoré takúto klasifikáciu nevieme aplikovať: sú to problémy, pre ktoré nie je známy polynomiálny algoritmus, ale zároveň ani o nich nevieme, či sú NP–úplné. Ako príklad môžeme uviesť problém grafového izomorfizmu (t.j. pre dané dva grafy rozhodnúť, či sú izomorfné alebo nie). Preto je prirodzenou otázka: *Existujú problémy, ktoré patria do* NP *ale pritom nepatria do* P *a ani nie sú* NP*–úplné?*

Možností je niekoľko. Prvá z nich, P = NP, je triviálna, ale najmenej pravdepodobná. Ak P = NP, tak každý problém z NP má polynomiálny algoritmus a súčasne je NP–úplný. Druhou možnosťou je, že P $\neq$ NP. Potom môže nastať situácia, keď každý problém z NP je buď z P alebo je NP–úplný. Možná je aj situácia, keď existuje jazyk z NP, ktorý ani nepatrí do P, ani nie je NP–úplný (pozri obrázok 6.1). Ukážeme, že pravdivé môžu byť len prvý a posledný prípad.



Obrázek 6.1: Vzťah tried P a NP

Veta 6.2 *Ak* P $\neq$ NP*, tak existuje jazyk z* NP*, ktorý ani nepatrí do* P*, ani nie je* NP*–úplný.*

Dôkaz Predpokladajme, že $\mathcal{M}_1, \mathcal{M}_2, \ldots$ je efektívne očíslovanie všetkých polynomiálne ohraničených Turingovych strojov. Pod pojmom efektívne očíslovanie pritom rozumieme, že existuje Turingov stroj, ktorý počíta jeden po druhom kódy $\mathcal{M}_1, \mathcal{M}_2, \ldots$ Existencia predpokladaného očíslovania sa ľahko overí: vyjdime z očíslovania $\overline{\mathcal{M}_1}, \overline{\mathcal{M}_2}, \ldots$ Turingovych strojov, ako sme ho prezentovali v časti 3.1. Skonštruujeme nové očíslovanie tak, že ku každému stroju pripojíme „vnútorné hodiny", ktoré ho zastavia po polynomiálnom počte krokov: $\forall i, j\ \mathcal{M}_{\langle i,j \rangle}$ [1] je stroj, ktorý na vstupe w simuluje $|\, w\, |^j$ krokov stroja $\overline{\mathcal{M}_i}$.

Podobným spôsobom môžeme predpokladať, že $\mathcal{R}_1, \mathcal{R}_2, \ldots$ je efektívne očíslovanie polynomiálne časovo ohraničených strojov, počítajúcich redukcie. Ďalej predpokladajme, že $\mathcal{S}$ je deterministický Turingov stroj, ktorý rozhoduje problém splniteľnosti CNF (keďže predpokladáme P $\neq$ NP, tak $\mathcal{S}$ nemá polynomiálnu časovú zložitosť.

[1] Symbolom $\langle\rangle$ tu označujeme párovaciu funkciu na množine prirodzených čísiel $\mathbb{N}$. Párovacia funkcia je pre dvojice prirodzených čísiel definovaná ako $\langle i, j \rangle = 1 + 2 + \cdots + (i + j) + j$.

Opíšeme jazyk L, ktorý ani nepatrí do P, ani nie je NP–úplný. Presnejšie, popíšeme Turingov stroj $\mathcal{K}$, ktorý ho rozhoduje. Stroj $\mathcal{K}$ je definovaný takto:

$$\underline{\textbf{if}}\quad \mathcal{S}(x) = ACCEPT\ \wedge\ f(|x|) \textit{je párne}\ \ \underline{\textbf{then}}\ \ \mathcal{K}(x) = ACCEPT$$
$$\underline{\textbf{else}}\ \ \mathcal{K}(x) = REJECT$$

Inak povedané, stroj $\mathcal{K}$ akceptuje vstupný reťazec x práve vtedy, ak x je kódom splniteľnej booleovskej formuly v konjunktívnom normálnom tvare a súčasne funkcia f nadobúda párnu hodnotu. Pritom f je celočíselná funkcia $f : \mathbb{N} \to \mathbb{N}$, ktorej argumentom je dĺžka vstupného reťazca x. Definícia funkcie f je jadrom dôkazu.

Definujeme funkciu f. Je to neklesajúca funkcia ($f(n+1) \geq f(n)$), ktorá rastie *veľmi* pomaly. Opíšeme Turingov stroj $\mathcal{F}$ so vstupnou páskou, ktorý počíta funkciu f; argument n funkcie f je zakódovaný ako vstupný reťazec 1^n stroja $\mathcal{F}$. Stroj $\mathcal{F}$ pracuje v dvoch etapách; v každej z nich vykoná presne n krokov. Technická realizácia: $\mathcal{F}$ posúva hlavu na vstupnej páske v každom kroku výpočtu o 1 políčko doprava; prečítanie prázdneho symbolu $\sqcup$ znamená ukončenie prvej etapy výpočtu. Počas druhej etapy výpočtu sa hlava na vstupnej páske posúva v každom kroku o 1 políčko doľava. Prečítanie symbolu $\triangleright$ znamená ukončenie druhej etapy výpočtu.

Počas prvej etapy výpočtu počíta $\mathcal{F}$ na svojej pracovnej páske postupne hodnoty $f(0), f(1), f(2)$ atď.; toľko, koľko ich je schopný za n krokov vypočítať. Nech posledná vypočítaná hodnota funkcie f bola $f(j) = k$. *Hodnota $f(n)$ bude rovná buď k alebo $k+1$.* O tom, ktorá z nich to bude, sa rozhodne v druhej etape výpočtu stroja $\mathcal{F}$.

Už teraz si môžeme overiť, že funkcia f rastie veľmi pomaly. Ak $n(k)$ je najmenšie číslo, pre ktoré $f(n) = k$, tak je zrejmé, že najmenšie číslo, pre ktoré môže $f(n)$ byť rovné $k+1$ je prinajmenšom $\frac{n(k)^2}{2}$. Pri tomto odhade vychádzame zo skutočnosti, že na výpočet hodnoty $f(l)$ je potrebných aspoň l krokov. Potom ale $f(n) = \mathcal{O}(\log\log n)$, čo je skutočne veľmi pomaly rastúca funkcia. Z toho, ako upresníme výpočet funckie f v druhej etape výpočtu stroja $\mathcal{F}$ vyplynie, že v skutočnosti rastie f ešte pomalšie.

Výpočet v druhej etape závisí od toho, či číslo k vypočítané v prvej etape je párne alebo nepárne. Uvážme najprv prípad, keď $k = 2i$ je párne. Stroj $\mathcal{F}$ začne simulovať výpočty $\mathcal{M}_i(z)$, $\mathcal{S}(z)$ a $\mathcal{F}(|z|)$, kde za z sa dosadzujú postupne binárne reťazce v rastúcom usporiadaní. Opäť $\mathcal{F}$ simuluje toľko výpočtov, koľko ich za n krokov môže. Pritom $\mathcal{F}$ sa snaží nájsť taký reťazec z, pre ktorý platí

$$\mathcal{K}(z) \quad \neq \quad \mathcal{M}_i(z) \tag{6.1}$$

kde $\mathcal{K}$ je vyššie popísaný stroj rozhodujúci jazyk L. Podľa definície stroja $\mathcal{K}$ to znamená, že $\mathcal{F}$ hľadá z také, že

- buď $\mathcal{M}_i(z) = ACCEPT$ a potom buď $\mathcal{S}(z) = REJECT$ alebo $f(|z|)$ je nepárne
- alebo $\mathcal{M}_i(z) = REJECT$ a $\mathcal{S}(z) = ACCEPT$ a $f(|z|)$ je párne.

Predpokladajme teraz, že $k = 2i-1$ je nepárne. Potom v druhej etape svojho výpočtu simuluje $\mathcal{F}$ výpočty $\mathcal{R}_i(z)$, $\mathcal{S}(\mathcal{R}_i(z))$ a $\mathcal{F}(|\mathcal{R}_i(z)|)$ ($\mathcal{R}_i$ je redukcia a jej výsledkom je reťazec). Opäť za z berie postupne všetky binárne reťazce v rastúcom usporiadaní. Tentokrát $\mathcal{F}$ hľadá taký reťazec z, pre ktorý platí

$$\mathcal{K}(\mathcal{R}_i(z)) \quad \neq \quad \mathcal{S}(z) \tag{6.2}$$

Teda pre reťazec z musí platiť

- buď $\mathcal{S}(z) = ACCEPT$ a potom buď $\mathcal{S}(R_i(z)) = REJECT$ alebo $f(|\mathcal{R}_i(z)|)$ je nepárne
- alebo $\mathcal{S}(z) = REJECT$ a $\mathcal{S}(\mathcal{R}_i(z)) = ACCEPT$ a $f(|\mathcal{R}_i(z)|)$ je párne.

Ak $\mathcal{F}$ takéto z nájde (v rámci limitu n krokov), tak $f(n) = k+1$, inak $f(n) = k$.

Tým sme ukončili definíciu stroja $\mathcal{F}$ počítajúceho funkciu f s časovou zložitosťou $\mathcal{O}(n)$. To ale implikuje, že aj stroj $\mathcal{K}$ je úplne definovaný a teda následne aj jazyk L. Ľahko nahliadneme, že $L \in$ NP. Pre vstup x stačí urobiť nasledovné dve akcie: Uhádnuť spĺňajúce priradenie hodnôt premenným a vypočítať $f(|x|)$. Oba výpočty sa dajú realizovať nedeterministicky v polynomiálnom čase.

Musíme ešte ukázať, že L ani nepatrí do P, ani nie je NP–úplný. Predpokladajme najprv, že $L \in$ P. Potom$L$ je rozhodovaný nejakým deterministickým polynomiálne časovo ohraničeným Turingovym strojom, v našom očíslovaní $\mathcal{M}_i$. To jest $L = L(\mathcal{M}_i)$, alebo inak zapísané, $\mathcal{K}(z) = \mathcal{M}_i(z)$ pre všetky z. Za predpokladu platnosti tohoto vzťahu , v druhej etape svojho výpočtu pre $k = 2i$ stroj $\mathcal{F}$ nikdy nenájde z, pre ktoré by bola splnená podmienka 6.1. Preto $f(n) = 2i$ pre všetky $n \geq n_0$, pre nejaké $n_0 \in \mathbb{N}$. Následne $f(n)$ je párne pre všetky, až na konečne veľa, prirodzené čísla. To znamená, že L koinciduje (je zhodné) s jazykom CNF pre všetky až na konečne veľa reťazcov. Tým sa ale dostávame do sporu s našimi dvoma predpokladmi: $L \in$ P a P $\neq$ NP. Preto $L \notin$ P.

Predpokladajme teda, že L je NP–úplný jazyk. Ľahko sa dostaneme k podobnému sporu. Keďže L je NP–úplný problém, tak existuje redukcia jazyka CNF na L, v našom očíslovaní R_i. Teda, $\mathcal{K}(\mathcal{R}_i(z)) = \mathcal{S}(z)$ pre všetky z. Za predpokladu platnosti tohoto vzťahu, v druhej etape svojho výpočtu pre $k = 2i - 1$ stroj $\mathcal{F}$ nemôže nájsť reťazec z spĺňajúci 6.2. Preto $f(n) = 2i - 1$ pre všetky, s výnimkou konečne veľa, prirodzené čísla. Ak spojíme tento fakt s popisom stroja $\mathcal{K}$ (rozhodujúceho jazyk L) vidíme, že L je konečný jazyk. Preto L patrí do P. Čo je spor s predpokladom , že L je NP–úplný a P $\neq$ NP. □

Najvážnejšími kandidátmi na príslušnosť do triedy jazykov, ktoré ani nepatria do P, ani nie sú NP–úplné sú problémy grafového izomorfizmu (pre dané dva grafy rozhodnúť, či sú izomorfné) a problém prvočíselnosti (o danom prirodzenom čísle rozhodnúť, či je prvočíslom).

Zovšeobecnením techniky uvedenej v dôkaze vety 6.2 sa dajú získať analogické výsledky aj pre iné zložitostné triedy.

6.3 Izomorfizmus NP–úplných jazykov

Všetky NP–úplné jazyky navzájom spolu veľmi úzko súvisia, pretože ľubovoľný z nich je možné redukovať na ktorýkoľvek iný. Ukážeme, že súvislosť medzi nimi je ešte tesnejšia a že všetky tieto jazyky sú „v podstate“ rovnaké, t.j. rovnaké až na polynomiálne vypočítateľnú permutáciu svojich prvkov. Formálne, všetky NP–úplné jazyky sú polynomiálne izomorfné.

Definícia 6.3 *Dva jazyky A, $B \subseteq \Sigma^*$ sú polynomiálne izomorfné, ak existuje funkcia $h : \Sigma^* \to \Sigma^*$ spĺňajúca:*

1. *h je bijekcia*
2. *pre každé $x \in \Sigma^*$ platí $x \in A \iff h(x) \in B$*
3. *obidve funkcie h a h^{-1} (pretože h je bijekcia, existuje k nej inverzná funkcia) sú vypočítateľné v polynomiálnom čase.*

Príklad 6.4 *Kandidátom na polynomiálny izomorfizmus je každá redukcia, ktorá je bijekciou. Nanešťastie väčšina redukcií, s ktorými sme sa stretli v kapitole 5, nie je bijektívna. Výnimkou je redukcia problému k–KLIKA na problém k–NEZÁVISLÁ MNOŽINA, ktorá prípadu (G, k) problému k–KLIKA priradí prípad (G^c, k) problému k–NEZÁVISLÁ MNOŽINA, kde G^c je graf komplementárny ku grafu G.*

Vo všeobecnosti, ak A a B sú dva NP–úplné jazyky, tak existujú redukcie f (redukuje A na B) a redukcia g (redukuje B na A). Avšak nemusí platiť $g = f^{-1}$ a preto z existencie f a g ešte nevyplýva, že jazyky A a B sú polynomiálne izomorfné. Existuje však jednoduchý, systematický spôsob, ako transformovať ľubovoľnú redukciu na bijektívnu redukciu. Je založený na pojmoch dĺžkou rastúca a dĺžkou klesajúca funkcia.

Definícia 6.5 *Funckia $f : \Sigma^* \to \Gamma^*$ je* dĺžkou rastúca *ak, pre každé $x \in \Sigma^*$ platí*

$$|f(x)| > |x|.$$

Funkcia je dĺžkou klesajúca *ak, pre každé $x \in \Sigma^*$ je*

$$|f(x)| < |x|.$$

Nasledujúca veta vyjadruje nutné podmienky existencie polynomiálneho izomorfizmu medzi dvoma jazykmi.

Lema 6.6 *Nech $A \subseteq \Sigma^*$ a $B \subseteq \Gamma^*$ sú jazyky. Nech $f : \Sigma^* \to \Gamma^*$ je redukcia z A do B a $g : \Gamma^* \to \Sigma^*$ redukcia z B do A. Nech obidve redukcie sú dĺžkou rastúce, vypočítateľné v polynomiálnom čase a invertovateľné*[2]*, pričom aj inverzné funkcie f^{-1} a g^{-1} sú vypočítateľné v polynomiálnom čase. Potom jazyky A a B sú polynomiálne izomorfné.*

Dôkaz Našim cieľom je, vychádzajúc z redukcií f a g, skonštruovať redukciu h, ktorá je bijekciou. Pre tento účel rozdeľme množiny Σ^* na podmnožiny R_1 a R_2 a Γ^* na podmnožiny S_1 a S_2 nasledovne (pozri obrazok 6.2). Pre dané slovo $x \in \Sigma^*$

$x \in R_1$ ak existuje celé číslo $k \geq 0$ také, že kompozícia $(f^{-1} \circ g^{-1})^k$ aplikovaná na x je definovaná, zatiaľ čo kompozícia $g^{-1} \circ (f^{-1} \circ g^{-1})^k$ je nedefinovaná,

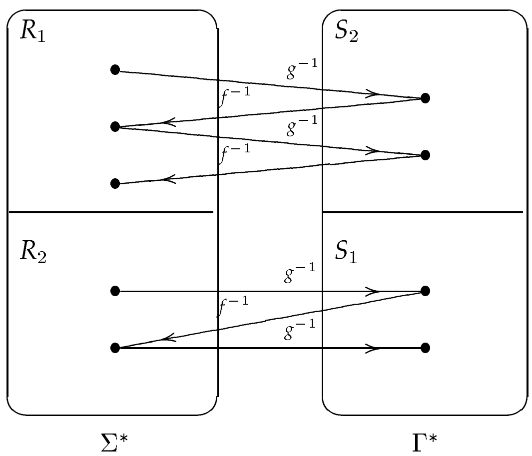
$x \in R_2$ ak existuje celé číslo $k \geq 0$ také, že kompozícia $g^{-1} \circ (f^{-1} \circ g^{-1})^k$ aplikovaná na x je definovaná, zatiaľ čo pre to isté x je kompozícia $(f^{-1} \circ g^{-1})^{k+1}$ nedefinovaná.

Inak povedané, slovo x patrí do R_1 (R_2) práve ak najdlhšia možná kompozícia inverzných funkcií f^{-1} a g^{-1} ho prevedie na slovo z Σ^* (Γ^*). Dôležité je uvedomiť si, že ak vyjdeme z R_1 (R_2), nikdy sa nedostaneme do R_2 (R_1).

Podobne definujeme rozdelenie Γ^* na množiny S_1 a S_2. Pre dané slovo $y \in \Gamma^*$

$y \in S_1$ ak existuje $k \geq 0$, že kompozícia $(g^{-1} \circ f^{-1})^k$ aplikovaná na y je definovaná, avšak kompozícia $f^{-1} \circ (g^{-1} \circ f^{-1})^k$ pre to isté y nie je definovaná,

$y \in S_2$ ak existuje $k \geq 0$, že kompozícia $f^{-1} \circ (g^{-1} \circ f^{-1})^k$ aplikovaná na y je definovaná, ale kompozícia $(g^{-1} \circ f^{-1})^{k+1}$ pre to isté y nie je definovaná.



Obrázek 6.2: Rozdelenie množín Σ^* a Γ^*

Keďže f a g sú dĺžkou rastúce, tak f^{-1} a g^{-1} sú dĺžkou klesajúce. Preto pre každé $x \in \Sigma^*$ vieme v polynomiálnom čase rozhodnúť, či patrí do R_1 alebo do R_2 (striedavo budeme počítať g^{-1} a f^{-1}, nanajvýš $|x|$–krát). Podobne pre každé $y \in \Gamma^*$ vieme v polynomiálnom čase rozhodnúť, či patrí do S_1 alebo do S_2.

Teraz už sme pripravení definovať funkciu h

$$h(x) \stackrel{\text{def}}{=} \begin{cases} f(x) & \text{ak } x \in R_1 \\ g^{-1}(x) & \text{ak } x \in R_2 \end{cases}$$

[2]Funkcia f je invertovateľná, ak k nej existuje inverzná funkcia, t.j ak f je injektívna

a k nej inverznú funkciu h^{-1}

$$h^{-1}(y) \stackrel{\text{def}}{=} \begin{cases} g(y) & \text{ak } y \in S_1 \\ f^{-1}(y) & \text{ak } y \in S_2 \end{cases}$$

Overíme, že h má požadované vlastnosti, tj. že h je redukcia, h aj h^{-1} sú vypočítateľné v polynomiálnom čase a h je bijekcia.

Podľa predpokladu je funkcia f resp. g redukciou z A do B, resp. z B do A. Preto pre každé x platí: $x \in A \iff h(x) \in B$. Naviac, pretože f, g, f^{-1} a g^{-1} sú polynomiálne vypočítateľné, tak aj h a h^{-1} sú polynomiálne vypočítateľné.

Nakoniec overíme bijektivitu, t.j. že pre každé $x \in \Sigma^*$ je $h^{-1}(h(x)) = x$ a pre každé $y \in \Gamma^*$ je $h(h^{-1}(y)) = y$. Vyjdeme z toho, že ak $x \in R_1$, tak $f(x) \in S_2$. Skutočne, zo vzťahu $f(x) \in S_1$ by vyplývala (na základe definície množiny S_1) existencia celého čísla $k \geq 0$, že $(g^{-1} \circ f^{-1})^k[f(x)$ je definované, ale $f^{-1} \circ (g^{-1} \circ f^{-1})^k[f(x)]$ je nedefinované. Z toho ale plynie (dosadením $f^{-1} \circ f[x] = x$), že kompozícia $g^{-1} \circ (f^{-1} \circ g^{-1})^{k-1}$ aplikovaná na to isté x je definovaná, ale $(f^{-1} \circ g^{-1})^k$ je nedefinovaná. To je ale spor s predpokladom, že $x \in R_1$. Pomocou rovnakých argumentov môžeme ukázať, že ak $x \in R_2$, tak $f(x) \in S_1$, ďalej že z $y \in S_1$ plynie $g(y) \in R_2$ a z $y \in S_2$ plynie $g(y) \in R_1$. Teraz už stačí len dať do súvislosti tieto vzťahy s definíciou funkcie h aby sme videli, že h je bijekcia. Preto A a B sý polynomiálne izomorfné. □

V nasledujúcej časti ukážeme, ako sa dá ľubovoľná redukcia transformovať na injektívnu, dĺžkou rastúcu funkciu, ktorej inverzná funkcia je polynomiálne časovo vypočítateľná. Spolu s lemou 6.6 nám tento návod poskytne algoritmus, ako redukciu transformovať na bijektívnu redukciu. Celý postup je založený na princípe tzv. *predlžovacej funkcie* (z angl. padding function).

Definícia 6.7 *Nech $L \in \Sigma^*$ je jazyk. Hovoríme, že funkcia $pad : (\Sigma^*)^2 \to \Sigma^*$ je* predlžovacia funkcia pre *L, ak sú splnené podmienky:*

1. *Funkcia pad je polynomiálne časovo vypočítateľná*
2. *Pre ľubovoľné $x, y \in \Sigma^*$ platí $pad(x, y) \in L \iff x \in L$*
3. *Pre ľubovoľné $x, y \in \Sigma^*$ platí $|\,pad(x, y)\,| > |\,x\,| + |\,y\,|$*
4. *Existuje polynomiálne časovo ohraničený algoritmus, ktorý pre danú hodnotu $pad(x, y)$ vypočíta y.*

Funkcia pad je v skutočnosti dĺžkou rastúca redukcia z L do L, ktorá slovu z L „prikóduje“ (pripojí) ďalší reťazec y.

Príklad 6.8 *Navrhneme predlžovaciu funkciu pre problém splniteľnosti CNF. Pre daný prípad x problému CNF—booleovskú formulu s n premennými—a reťazec y definujeme funkciu pad takto: $pad(x, y)$ bude boolevská formula obsahujúca formulu x a naviac ďalších $|y| + 1$ premenných a $|y| + 1$ klauzulí:*

$$pad(x, y) \stackrel{\text{def}}{=} x \wedge (x_{n+1} \vee \neg x_{n+1}) \wedge l_1 \wedge \ldots \wedge l_{|y|},$$

kde

$$l_i \stackrel{\text{def}}{=} \begin{cases} x_{n+i+1} & \text{ak } i\text{–ty bit reťazca } y \text{ je 1} \\ \neg x_{n+i+1} & \text{ak } i\text{–ty bit reťazca } y \text{ je 0.} \end{cases}$$

Bez straty všeobecnosti predpokladáme, že Σ je $\{0, 1\}$.

Tvrdíme, že takto definovaná funkcia pad je skutočne predlžovacia funkcia pre jazyk CNF. Zrejme funkcia pad je vypočítateľná v polynomiálnom čase. Ďalej, pridanie nových klauzúl nijako neovplyvní splniteľnosť. Funkcia je dĺžkou rastúca. Navyše, pre dané $pad(x, y)$ ľahko identifikujeme, kde končí pôvodná formula x a dokážeme zrekonštruovať y.

Príklad 6.9 *Skonštruujeme predlžovaciu funkciu pre problém k–VRCHOLOVÉ POKRYTIE. Pre daný graf $G = (V, H)$ s množinou vrcholov $\{x_1, \ldots, x_n\}$ a pre reťazec $y = y_1 \ldots y_q$ definujeme*

$$pad(G, y) \stackrel{\text{def}}{=} G' = (V', H'),$$

kde V' je rozšírením V o $2q + 3$ nových vrcholov $x_{n+1}, \ldots, x_{n+2q+3}$ a
$H' = H \cup \{(x_{n+1}, x_{n+2}), (x_{n+1}, x_{n+3}), (x_{n+2}, x_{n+3})\} \cup \{\, (x_{n+2i+2}, x_{n+2i+3}) \mid y_i = 1 \,\}$.
Graf G má pokrytie veľkosti nanajvýš k práve ak G' má pokrytie veľkosti nanajvýš $k + 2 + n_1$, kde n_1 je počet jednotiek v reťazci y. Podobne ako v predchádzajúcom príklade, z daného $pad(G, y)$ ľahko zrekonštruujeme y.

Pre väčšinu známych NP–úplných problémov sa predlžovacia funkcia konštruuje veľmi jednoducho. Pomocou nej dokážeme transformovať redukciu na injektívnu redukciu.

Lema 6.10 *Nech $A, B \subseteq \Sigma^*$ sú jazyky a f je polynomiálna redukcia z A do B. Nech pad je predlžovacia funkcia pre jazyk B.*
Potom funkcia, zobrazujúca $x \in \Sigma^*$ na $pad(f(x), x)$ je dĺžkou rastúca, invertovateľná redukcia z A do B. Naviac, k nej inverzná funkcia (pre dané $pad(f(x), x)$ vypočíta x) je vypočítateľná v polynomiálnom čase.

Dôkaz Skutočnosť, že $pad(f(x), x)$ je redukcia vyplýva z faktu, že f je redukcia a z vlastností 1 a 2 predlžovacej funkcie. Vlastnosť 3 predlžovacej funkcie implikuje, že skonštruovaná redukcia je dĺžkou rastúca. Konečne, vlastnosť 4 zaručuje, že aj inverzná funkcia je polynomiálne vypočítateľná. □

Skombinovaním výsledkov prezentovaných v lemách 6.6 a 6.10 dostávame nástroj na dokazovanie polynomiálneho izomorfizmu medzi NP–úplnými problémami: dva NP–úplné jazyky A a B sú polynomiálne izomorfné, ak pre ne existujú predlžovacie funkcie.

Príklad 6.11 *Problémy CNF a k–VRCHOLOVÉ POKRYTIE sú polynomiálne izomorfné, pretože (ako ukazujú predchádzajúce dva príklady) pre každý z nich existuje predlžovacia funkcia.*

Keďže ku každému doteraz známemu NP–úplnému problému sa podarilo nájsť predlžovaciu funkciu, sú všetky známe NP–úplné problémy polynomiálne izomorfné. Táto skutočnosť viedla Hartmanisa a Bermana k formulovanie tzv. *hypotézy o p–izomorfizme*:

> *Všetky* NP*–úplné jazyky sú polynomiálne izomorfné.*

Nasledujúca veta ukazuje, že dokázať práve uvedenú hypotézu nemusí byť vôbec jednoduché.

Veta 6.12 *Ak všetky* NP*–úplné jazyky sú navzájom polynomiálne izomorfné, tak* $\mathsf{P} \neq \mathsf{NP}$.

Dôkaz Predpokladajme, že $\mathsf{P} = \mathsf{NP}$. Potom každý neprázdny konečný jazyk patrí do P a je NP–úplný (dokážte). Ale konečný jazyk nemôže byť polynomiálne izomorfný s nekonečným jazykom (napr. s jazykom CNF). □

V skutočnosti hypotéza o p–izomorfizme nemusí byť pravdivá a môže sa stať, že existuje NP–úplný jazyk, ktorý nie je polynomiálne izomorfný s jazykom CNF.

6.4 NP–úplnosť a riedke jazyky

Polynomiálny izomorfizmus úzko súvisí s ďalšou dôležitou vlastnosťou jazykov a to s ich hustotou. Nech $L \subseteq \Sigma^*$ je jazyk. Jeho *hustota* je funkcia $dens_L$ definovaná na množine prirodzených čísiel:

$$dens_L(n) \stackrel{\text{def}}{=} |\{\, x \in L \mid |x| \leq n \,\}|.$$

Inak povedané, $dens_L(n)$ je rovné počtu slov jazyka L, ktorých dĺžka nepresahuje n. Hustota jazyka nemôže rásť rýchlejšie než exponenciálna funkcia. Z toho vyplýva rozlíšenie jazykov na *riedke jazyky*, ktorých hustota je polynomiálne ohraničena a na *husté jazyky* so superpolynomiálnou hustotou. Príkladom riedkych jazykov sú tzv. unárne jazyky, tj. jazyky definované nad jednoprvkovou abecedou. Evidentne je hustota ľubovoľného unárneho jazyka U $dens_U(n) \leq n$.

Vzťah medzi hustotou a polynomiálnym izomorfizmom je zachytený v nasledujúcej leme.

Lema 6.13 *Ak $A, B \subseteq \Sigma^*$ sú polynomiálne izomorfné jazyky, tak funkcie $dens_A$ a $dens_B$ sú v polynomiálnom vzťahu.*

Dôkaz Všetky reťazce jazyka A dĺžky nepresahujúcej n sa polynomiálnym izomorfizmom zobrazia na reťazce jazyka B dĺžky nepresahujúcej $p(n)$, kde izomorfizmus je vypočítateľný v čase ohraničenom polynómom p. Tento izomorfizmus je injektívny, preto $dens_A(n) \leq dens_B(p(n))$. Analogicky dostaneme $dens_B(n) \leq dens_A(p'(n))$, kde p' je polynóm ohraničujúci časovú náročnosť výpočtu inverzného izomorfizmu. □

Všetky známe NP–úplné jazyky sú husté a preto podľa lemy 6.13 nemôžu byť izomorfné s riedkym jazykom. Z platnosti hypotézy o p–izomorfizme by preto vyplývalo, že neexistuje riedky NP–úplný jazyk. Naopak, ak by sa podarilo nájsť riedky NP–úplný problém, znamenalo by to vyvrátenie hypotézy o p–izomorfizme.

Otázka existencie riedkeho NP–úplného problému je zaujímavá aj z praktického hľadiska. Predpokladajme, že máme riedky NP–úplný problém R a nech L je ľubovoľný jazyk z triedy NP. Z NP–úplnosti jazyka R vyplýva existencia redukcie z L do R vypočítateľnej v polynomiálnom čase $p(n)$. Potom pre ľubovoľné dané n vieme skonštruovať tabuľku, obsahujúcu všetky slová jazyka R dĺžky nepresahujúcej $p(n)$. Veľkosť tejto tabuľky je zhora ohraničená polynómom $dens_R(p(n))$. Konštrukcia tabuľky si vyžaduje exponenciálny čas, pretože pre exponenciálne veľa slov musíme overiť, či patria do R. Ak však už máme tabuľku vypočítanú, dokážeme v polynomiálnom čase rozhodovať o ľubovoľnom slove dĺžky nepresahujúcej n, či patrí do jazyka L. Stačí vypočítať redukciu tohoto slova (polynomiálny čas) a overiť, či redukcia patrí do už skonštruovanej tabuľky (polynomiálnej veľkosti).

Nasledujúci výsledok ukazuje, že za predpokladu P $\neq$ NP, žiaden NP–úplný jazyk nemôže byť riedky.

Veta 6.14 *Ak existuje riedky* NP*–úplný jazyk, tak* P $=$ NP.

Dôkaz Načrtneme hlavné myšlienky dôkazu. Pozostáva z dvoch častí. Najprv definujeme NP–úplný jazyk A. V druhej časti, využijúc predpokladanú existenciu riedkeho NP–úplného jazyka, skonštruujeme deterministický, polynomiálne časovo ohraničený Turingov stroj rozhodujúci A. Teda $A \in$ P a P $=$ NP.

Jazyk A definujeme ako

$$A \stackrel{\text{def}}{=} \{ \langle x, F \rangle \mid \begin{array}{l} F \text{ je booleovská formula } r \text{ premenných,} \\ x \in \{0,1\}^r \\ \exists y : (x < y \wedge F(y) = 1) \, \}. \end{array}$$

Zrejme jazyk A patrí do NP. Zároveň je NP–ťažký, pretože CNF $\leq A$ (redukcia priradí formule F slovo $\langle 00 \ldots 0, F \rangle$).

Nech R je riedky NP–úplný jazyk, ktorého hustota $dens_R$ je ohraničená polynómom n^k. Ďalej nech f je polynomiálne časovo vypočítateľná funkcia, ktorá je redukciou z A do R a pre ktorú $|f(x)| \leq N(|x|)$ (N polynóm). Funkcia f existuje, pretože R je NP–úplný a $A \in$NP.

Majme dané slovo $\langle x, F \rangle$ dĺžky n, kde F je formula r premenných a $r \in \{0,1\}^r$, o ktorom chceme rozhodnúť, či patrí do jazyka A. Urobíme to takto: nájdeme (ak existuje) najväčší binárny reťazec $\overline{y}$ dĺžky r taký, že $\langle \overline{y}, F \rangle \in A$. Ak ho nájdeme, stačí porovnať reťazce x a $\overline{y}$ a vieme, či $\langle x, F \rangle \in A$.

Predstavme si 2^r binárnych reťazcov dĺžky r zapísaných za sebou do jedného riadku v lexikografickom usporiadaní. Rozdeľme tento riadok na $2(N(n))^k$ segmentov a vypočítajme hodnotu funkcie f (f je redukcia z A do R) pre $2(N(n))^k$ vektorov zo začiatku každého segmentu. Predpokladajme, že všetkých $2(N(n))^k$ vypočítaných hodnôt je rôznych. Potom medzi prvými $(N(n))^k + 1$ spomedzi vypočítaných hodnot musí byť taká hodnota $f(\langle y, F \rangle)$, ktorá nepatrí do jazyka R (každá vypočítaná hodnota je reťazec dĺžky nepresahujúcej $N(n)$ a takýchto reťazcov obsahuje riedky jazyk R nanajvýš $(N(n))^k$). To ale znamená, že hľadané $\overline{y}$ určite neleží vpravo od y. Preto môžeme odstrániť pravú polovičku riadku a hľadať riešenie len v jej ľavej polovičke. Celý postup teraz opakujeme pre novovytvorený riadok (polovičnej dĺžky). Zrejme ak aj naďalej budeme mať šťastie, a vypočítané hodnoty funkcie f budú vždy rôzne, tak po polynomiálnom počte opakovaní sa dostaneme k polynomiálnemu počtu hodnôt, mezi ktorými musí byť—ak existuje—aj hľadané spĺňajúce priradenie $\overline{y}$. Preto stačí preveriť každé z nich a celý problém sme vyriešili v polynomiálnom čase.

Samozrejme musíme uvážiť aj situáciu, keď nie všetky vypočítané hodnoty funkcie f sú navzájom rozdielne. Využijeme to, že aj dve rovnaké hodnoty obsahujú dostatok informácie. Môžeme totižto odstrániť všetky segmenty nachádzajúce sa medzi bodmi s rovnakou hodnotou. Presnejšie: ak $y < y'$ a pritom $f(\langle y, F\rangle) = f(\langle y', F\rangle) \notin R$, tak hľadané $\overline{y}$ určite neleží napravo od y a teda ani medzi y a y'. Z druhej strany, ak $f(\langle y, F\rangle) = f(\langle y', F\rangle) \in R$, tak hľadané $\overline{y}$ leží napravo od y' a preto môžeme odstrániť segmenty medzi y a y' bez toho, aby sme pritom stratili hľadané riešenie. Ak medzi vypočítanými hodnotami je viac navzájom rovnakých, môžeme odstrániť segmenty medzi každou dvojicou bodov s rovnakými hodnotami. Ľahko nahliadneme, že skombinovaním oboch popísaných metód dokážeme vždy skrátiť riadok tak, že počet segmentov, medzi ktorými musíme hľadať riešenie, sa zníži aspoň o polovicu.

Tým sme dokázali, že $A \in \mathsf{P}$, z čoho už plynie $\mathsf{P} = \mathsf{NP}$. □

6.5 Relativizácia $\mathsf{P} \stackrel{?}{=} \mathsf{NP}$ problému

Až doteraz sme sa zaoberali v podstate len dvoma zložitostnými triedami, P a NP, a vzťahom medzi nimi. Videli sme, že napriek snahe pristupovať k otázke ich vzťahu z rôznych strán, nebola doteraz nájdená uspokojivá odpoveď. To všetko viedlo k snahe nájsť odpoveď aspoň na takúto otázku: aký bude vzťah skúmaných tried P a NP, ak trochu zmeníme podmienky?

Zodpovedzme najprv otázku, čo rozumieme, v kontexte výpočtovej zložitosti, zmenou podmienok. Spoločnou charakterisktikou doteraz uvažovaných výpočtov je, že do celkovej zložitosti sa započítava každý vykonaný krok. Skúsme si predstaviť svet, v ktorom budú niektoré výpočty „zadarmo“. Napríklad, majme algoritmus, v ktorom keď počas výpočtu potrebujeme o nejakej booleovskej formule rozhodnúť, či je splniteľná, dostaneme túto informáciu okamžite „zvonka“. Na základe poskytnutej odpovede potom výpočet pokračuje ďalej. Inak povedané, pohybujeme sa vo svete CNF, kde ľubovoľný dotaz na príslušnosť formuly do jazyka CNF je zodpovedaný okamžite a správne nejakým orákulom.

Samozrejme, že takýto svet má ďaleko od reality; uvedomme si, že slúži len ako prostriedok na skúmanie prirodzených problémov. Akonáhle máme totižto skonštruovaný tento alternatívny svet, môžeme sa pýtať, či v ňom platí $\mathsf{P} = \mathsf{NP}$ alebo nie. Definujeme teraz presne pojmy, s ktorými sme doteraz pracovali na intuitívnej úrovni. Potom sa znovu vrátime k otázke, čo nám takto získané výsledky môžu povedať o vzťahu tried P a NP.

Formalizáciou spomínaných pojmov je *orákulový Turingov stroj*.

Definícia 6.15 *Turingov stroj s orákulom $\mathcal{M}^?$ je viacpáskový deterministický Turingov stroj, ktorý má špeciálnu pásku (nazýva sa aj* dotazovacia páska*; $\mathcal{M}^?$ na ňu môže len zapisovať) a tri špeciálne stavy: $q_?$ (tzv.* dotazovací stav*) a q_{yes}, q_{no} (tzv.* odpovedacie stavy*).*

Nech $A \subseteq \Sigma^*$ je ľubovoľný jazyk. Výpočet orákulového Turingovho stroja $\mathcal{M}^?$ s orákulom A prebieha presne ako výpočet Turingovho stroja s výnimkou konfigurácie, keď $\mathcal{M}^?$ je v dotazovacom stave $q_?$. V nasledujúcom kroku výpočtu prejde $\mathcal{M}^?$ buď do stavu q_{yes} alebo do stavu q_{no} v závislosti od toho, či reťazec práve zapísaný na dotazovacej páske patrí do jazyka A alebo nie. Pritom sa celý obsah dotazovacej pásky vymaže. Prostredníctvom odpovedacieho stavu teda získava $\mathcal{M}^?$ informáciu o príslušnosti reťazca zapísaného na dotazovacej páske do jazyka A. Výsledok výpočtu orákulového stroja $\mathcal{M}^?$ s orákulom A na vstupe w označujeme ako $\mathcal{M}^A(w)$.

Časová zložitosť orákulového Turingovho stroja sa definuje presne tak isto, ako pre obecný Turingov stroj; každý dotazovací krok výpočtu sa ráta za jeden krok presne ako obyčajný krok. Spôsob, ako definovať priestorovú zložitosť nie je celkom jednoznačný; bližšie sa k tejto otázke vrátime na zodpovedajúcom mieste. Definíciu orákulového stroja je možné rozšíriť aj na nedeterministické Turingove stroje. Prostredníctvom orákulových strojov môžeme definovať zložitostné triedy.

Definícia 6.16 *Nech A je jazyk. Definujeme* $(\mathsf{DTIME}(f(n)))^A$ *ako triedu jazykov*

$$(\mathsf{DTIME}(f(n)))^A \overset{\text{def}}{=} \{ L \mid \text{L je rozhodovaný deterministickým orákulovým TS s orákulom A a s časovou zložitosťou } f(n) \}.$$

Analogicky pre nedeterministické triedy. Definíciu môžeme zobecniť pre prípad, keď A nie je jazyk, ale trieda jazykov—v takomto prípade požadujeme, aby L bol rozhodovaný orákulovým strojom s orákulom z triedy A.

Tým sme upresnili pojem nášho alternatívneho, relativizovaného sveta. Našou ďalšou snahou bude určiť vzťah relativizovaných tried P a NP. Motiváciou je skutočnosť, že na otázku $\mathsf{P} \overset{?}{=} \mathsf{NP}$ sa dá pozerať ako na špeciálny prípad otázky $\mathsf{P}^X \overset{?}{=} \mathsf{NP}^X$ pre $X = \emptyset$. Ukážeme prekvapujúci fakt, a to, že je možné skonštruovať dva rôzne relatizované svety, v ktorých má táto otázka dve rôzne odpovede. Výsledok tohoto typu sa nazýva *protichodná relativizácia*.

Veta 6.17 *Existuje orákulum A, pre ktoré* $\mathsf{P}^A = \mathsf{NP}^A$.

Veta 6.18 *Existuje orákulum B, pre ktoré* $\mathsf{P}^B \neq \mathsf{NP}^B$.

Dôkaz [vety 6.17] Našou snahou je nájsť taký jazyk A, ktorý eliminuje silu nedeterminizmu. Alebo, inak povedané, jazyk, ktorého pomocou sa deterministický výpočet stane rovnako silným ako nedeterministický výpočet. S takouto situáciou sme sa už stretli: podľa Savitchovej teorémy (veta 4.17) vieme, že triedy polynomiálne priestorovo ohraničených deterministických a nedeterministických výpočtov koincidujú.

Zvoľme A ako ľubovoľný PSPACE–úplný jazyk. Ukážeme

$$\mathsf{PSPACE} \subseteq \mathsf{P}^A \subseteq \mathsf{NP}^A \subseteq \mathsf{NPSPACE} \subseteq \mathsf{PSPACE}.$$

Platnosť prvej inklúzie vyplýva z PSPACE–úplnosti jazyka A: každý jazyk L z PSPACE sa dá rozhodovať polynomiálne časovo ohraničeným deterministickým orákulovým strojom, ktorý najprv vypočíta redukciu z jazyka L do A a potom urobí jediný dotaz na orákulum A. Druhá inklúzia je triviálna. K tretej inklúzii: každý nedeterministický polynomiálne časovo ohraničený orákulový Turingov stroj sa dá simulovať polynomiálne priestorovo ohraničeným strojom bez orákula tak, že namiesto dotazu na orákulum stroj priamo rozhodne (v polynomiálnom priestore) o príslušnosti dotazovaného slova do jazyka A. Štvrtá inklúzia je obsahom Savitchovej teorémy. Preto je $\mathsf{P}^A = \mathsf{NP}^A$. □

Argumenty ukazujúce platnosť druhej vety sú o niečo komplikovanejšie.

Dôkaz [vety 6.18] Tentokrát potrebujeme nájsť orákulum B, ktoré umocní silu nedeterminizmu a jazyk $L \in \mathsf{NP}^B - \mathsf{P}^B$. Najprv definujeme unárny jazyk L ako

$$L \overset{\text{def}}{=} \{0^n \mid \text{existuje } x \in B \text{ pre ktoré } |x| = n\}.$$

Platnosť $L \in \mathsf{NP}^B$ je zrejmá: nedeterministický stroj s orákulom B uhádne pre vstup 0^n reťazec x dĺžky n a použitím orákula overí, či $x \in B$.

Zostáva definovať orákulum B tak, aby $L \notin \mathsf{P}^B$. Použijeme pritom diagonalizáciu cez všetky polynomiálne časovo ohraničené orákulové deterministické Turingove stroje. Predpokladajme, že $\mathcal{M}_1^?, \mathcal{M}_2^?, \dots$ je očíslovanie všetkých takýchto strojov[3]. Predpokladáme, že každý stroj sa v tomto očíslovaní vyskytuje nekonečne veľa krát. Táto požiadavka je triviálne splnená, pretože ku každému stroju možno ľubovoľne veľa rôznymi spôsobmi pridať nové, nedosiahnuteľné stavy, ktoré nijako nezmenia jazyk rozhodovaný strojom. Pritom všetky tieto varianty sa musia niekde v očíslovaní objaviť.

Jazyk B budeme definovať postupne v etapách, v i–tej etape skonštruujeme množimu B_i, čo je množina všetkých slov jazyka B dĺžky nanajvýš i. Jazyk $B = \bigcup_{i=0}^{\infty} B_i$. Na začiatku i–tej

[3]Konštruciu hľadaného očíslovania možno nájsť napríklad v dôkaze Vety 6.2.

etapy máme už vypočítané B_{i-1}, množinu všetkých slov jazyka B dĺžky menšej než i. Naviac máme množinu $X \subseteq \Sigma^*$ *výnimiek*, reťazcov, ktoré nesmieme zaradiť do B. Na začiatku prvej etapy je $B_0 = X = \emptyset$. Popíšeme, ako pomocou nich vypočítame množinu B_i. Kritérium, na základe ktorého rozhodneme, ktoré slová dĺžky i patria do množimy B_i vyplýva z nášho cieľa skonštruovať B tak, aby $L \notin \mathsf{P}^B$: slovo 0^i musí rozlíšiť jazyk L a jazyk rozhodovaný strojom $\mathcal{M}_i^B$. Preto budeme simulovať $i^{\log i}$ krokov stroja $\mathcal{M}_i^B$ na vstupe 0^i. Upozorňujeme na výber počtu simulovaných krokov: je to funkcia menšia než exponenciálna, ale pritom rastúca asymptoticky rýchlejšie než každý polynóm.

Počas simulácie stroja $\mathcal{M}_i^B$ na vstupe 0^i môže orákulový stroj vzniesť dotaz „je $x \in B$?". Ako zodpovieme tento dotaz? Ak $|x| < i$, tak stačí ak zistíme, či $x \in B_{i-1}$. Ak $x \in B_{i-1}$, zodpovieme orákulový dotaz kladne, t.j. $\mathcal{M}_i^B$ prejde do stavu q_{yes}, inak prejde do stavu q_{no}. Ak ale $|x| \geq i$, tak $\mathcal{M}_i^B$ prejde do stavu q_{no} a naviac pridáme x do množiny X, aby sme si zapamätali naše rozhodnutie, že $x \notin B$.

Predpokladajme, že po nanajvýš $i^{\log i}$ krokoch simulácie sa výpočet zastaví v zamietajúcom stave. Pripomíname, že našim cieľom je zabezpečiť, aby $\mathcal{M}_i^B$ nerozhodoval jazyk L. Preto definujeme

$$B_i \stackrel{\text{def}}{=} B_{i-1} \cup \{\, x \in \{0,1\}^* \mid |x| = i;\;\; x \notin X \,\}\,.$$

Tým dosiahneme, že $0^i \in L$ (lebo B obsahuje slovo dĺžky i), ale pritom $\mathcal{M}_i^B$ nerozhoduje jazyk L (stroj $\mathcal{M}_i^B$ zamietol vstup 0^i). Potrebujeme ale ešte ukázať, že množina

$$\{\, x \in \{0,1\}^* \mid |x| = i;\;\; x \notin X \,\}$$

je naozaj neprázdna. Je to dané tým, že X zaručene neobsahuje viac než $\sum_{j=1}^{i} j^{\log j}$ reťazcov dĺžky i (to je celkový počet doteraz odsimulovaných krokov na všetkých strojoch a v každom kroku sa do X pridá nanajvýš 1 prvok); táto suma je ale menšia než 2^i, čo je počet reťazcov v $\{0,1\}^i$.

Ak $\mathcal{M}_i^B$ akceptuje vstup 0^i v uvažovanom čase $i^{\log i}$, tak kladieme $B_i \stackrel{\text{def}}{=} B_{i-1}$. Tak zabezpečíme, že $0^i \notin L$, a teda $L(\mathcal{M}_i^B) \neq L$.

Čo urobíme v prípade, že simulovaný výpočet stroja $\mathcal{M}_i^B$ na vstupe 0^i sa po $i^{\log i}$ krokoch nezastaví? Takáto situácia môže vzniknúť, pretože polynóm p, ohraničujúci časovú zložitosť stroja $\mathcal{M}_i^B$, môže byť tak veľký, a index i tak malý, že $i^{\log i} < p(i)$. V takomto prípade kladieme $B_i = B_{i-1}$, ako keby stroj vstup akceptoval. Tak ale nemáme zabezpečné, že $L(\mathcal{M}_i^B) \neq L$. Kľúčovým je tu predpoklad, že stroje ekvivalentné stroju $\mathcal{M}_i^B$ sa v zvolenom očíslovaní vyskytnú nekonečne veľa krát. Preto sa tento stroj objaví ako nejaké $\mathcal{M}_I^?$, kde I je také veľké, že $I^{\log I} \geq p(I)$. Tam bude zabezpečené, že $L(\mathcal{M}_i^B) = L(\mathcal{M}_I^B) \neq L$.

Tým sme ukončili definíciu orákula B. Konštrukcia bola prevedená tak, aby žiaden deterministický, polynomiálne časovo ohraničený orákulový stroj s orákulom B neakceptoval jazyk L. To dokazuje tvrdenie vety. □

Výsledok o protichodnej relativizácii problému naznačuje, že $\mathsf{P} \stackrel{?}{=} \mathsf{NP}$ je v skutočnosti veľmi ťažký problém. Obvyklá vzájomná simulácia ako prostriedok na dokázanie rovnosti dvoch zložitostných tried, ani diagonalizácia ako metóda na dokázanie nerovnosti medzi zložitostnými triedami nemôžu vyriešiť $\mathsf{P} \stackrel{?}{=} \mathsf{NP}$ problém. Obidve tieto techniky sú totižto použiteľné aj v relativizovanom svete (relativizujú). Predpokladajme, že chceme dokázať inklúziu $\mathsf{P} \subset \mathsf{NP}$ diagonalizačnou technikou, ktorou skonštruujeme jazyk L patriaci do $\mathsf{NP} - \mathsf{P}$. Tento istý postup sa ale dá priamo aplikovať aj na separáciu zodpovedajúcich relativizovaných tried, pre *ľubovoľné* orákulum. Teda, ak vieme separovať triedy P a NP diagonalizačnou technikou, tak vieme dokázať aj separáciu P^R a NP^R pre ľubovoľné orákulum R, čo je spor s vetou 6.17. Naopak, simulácia sa zdá byť najlepším spôsobom, ako dokázať, že $\mathsf{P} = \mathsf{NP}$. Znamená to, že potrebujeme obecnú metódu na simuláciu nedeterministického polynomiálne časovo ohraničeného algoritmu deterministickým polynomiálne čaosovo ohraničeným algoritmom. Opäť sa zdá, že simulačná technika sa dá použiť aj pre relativizované triedy. Takýto výsledok by viedol k sporu s vetou 6.18.

Obecne platí, že žiadna dôkazová technika, ktorá relativizuje, nemôže byť použitá na určenie vzťahu medzi dvoma zložitostnými triedami s protichodnou relativizáciou.

Výsledok o protichodnej relativizácii $\mathsf{P} \stackrel{?}{=} \mathsf{NP}$ problému dokázali v roku 1975 Baker, Gill a Solovay. Odvtedy boli dokázané v teórii zložitosti stovky výsledkov. S výnimkou niekoľko málo tvrdení, týkajúcich sa napríklad interaktívnych protokolov a triedy PSPACE, všetky relativizujú. Význam skúmania orákul a relativizovaných svetov spočíva v tom, že odhaľujú techniky, ktoré sa pravdepodobne nedajú použiť na riešenie daných problémov.

6.6 $\mathsf{P} \stackrel{?}{=} \mathsf{NP}$

Pokúsime sa analyzovať dôsledky rovnosti tried P a NP.

Napriek tomu, že o triede P sa zvykne hovoriť ako o triede prakticky riešiteľných problémov, tak P obsahuje množstvo „prirodzených" problémov, ktoré sa prakticky pomocou počítačov riešiť nedajú. Dôvodov je niekoľko.

Prvá príčina je vo vysokom stupni polynómu. Zatiaľ čo je pravdou, že veľká väčšina existujúcich polynomiálnych algoritmov má nízky stupeň polynómu (nie väčší než 5 alebo 6), z vety o časovej hierarchii vyplýva existenca problémov, ktoré sa nedajú riešiť v čase lepšom než $\mathcal{O}(n^k)$, pre ľubovoľne veľké k.

Príklad 6.19 *Uvažujme modifikáciu problému k-KLIKA, kde vstupom je graf a úlohou je určiť, či obsahuje k-kliku (pre pevné k). Jediný známy algoritmus na riešenie tohoto problému spočíva v prehľadávaní všetkých k prvkových podmnožín množiny vrcholov a overovaní, či indukujú úplný podgraf. Časová zložitosť algoritmu je úmerná počtu podmnožín, ktoré musíme preveriť, a teda $\mathcal{O}(n^k)$, kde n je počet vrcholov grafu. Algoritmus má teda polynomiálnu zložitosť, ale pre veľké hodnoty k je prakticky nepoužiteľný.*

Druhá prekážka môže byť v príliš vysokej hodnote konštanty. Pripomeňme si, že počet krokov polynomiálneho algoritmu je ohraničný hodnotou cn^k, kde c a k sú konštanty. Opäť pre veľkú väčšinu prakticky používaných algoritmov je konštanta c nízka a dominantným výrazom určujúcim zložitosť je polynóm n^k. Avšak nemôžeme *a priori* vylúčiť existenciu algoritmov s vysokými konštantami.

Tretí, a asi najzávažnejší, argument je nasledujúci. Vieme, že trieda P bola definovaná ako trieda jazykov (problémov), pre ktoré *existuje* polnynomiálny algoritmus na ich rozhodovanie. Vo väčšine prípadov sa dá príslušnosť jazyka do triedy P dokázať konštruktívne. Otázkou je, či môže nastať situácia, že vieme o existencii polynomiálne časovo ohraničeného Turingovho stroja pre nejaký konkrétny jazyk, ale nie sme schopní ho skonštruovať. Inými slovami: existujú jazyky, pre ktoré sa dá príslušnosť do triedy P dokázať *len* nekonštruktívnym spôsobom?

Odpoveď je áno. Nebudeme prezentovať výsledky, ktoré viedli k tomuto prekvapujúcemu zisteniu, pretože sú pomerne dlhé a technicky obtiažne. Pre ilustráciu ukážeme jeden problém uvedených vlastností, tj. problém, pre ktorý je možné dokázať jeho príslušnosť do triedy P, ale pritom nie je známy žiaden algoritmus (bez ohľadu na zložitosť) na jeho riešenie.

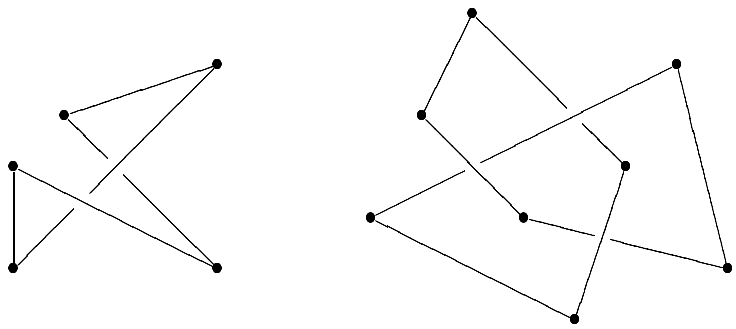
Príklad 6.20 *Uvažujem problém vnorenia grafu do trojrozmerného priestoru R^3. Je to priradenie, ktoré s každým vrcholom grafu asociuje bod a každej hrane grafu, spájajúcej dva vrcholy, priradí priamku, spájajúcu zodpovedajúce body. Naviac, priamky nesmú mať spoločné body, s výnimkou tých, ktoré sú asociované incidentným hranám a ktoré majú práve jeden spoločný bod.*

Predstavme si vnorenie grafu G do R^3; každý cyklus v G sa stane uzavretou čiarou v R^3. Neformálne, uzavretá čiara tvorí uzol, ak nie je možné natiahnuť ju do „kruhu" (ten sa nazýva aj triviálny uzol). Obrázok 6.20 ilustruje dve vnorenia, ktoré tvoria triviálny a netriviálny uzol.

Problém určiť pre daný graf, či je možné ho vnoriť do R^3 tak, aby vnorenie neobsahovalo uzly, je hľadaným problémom.

6.7 Funkcionálna verzia $\mathsf{P} \stackrel{?}{=} \mathsf{NP}$ problému

Až doteraz sme sa zaoberali iba *jazykmi* kódujúcimi *rozhodovacie problémy*. Otázkou je, či takéto zúženie pohľadu je opodstatnené. Výpočtové úlohy, ktoré potrebujeme v praxi riešiť, sú málokedy formulované ako úlohy, kde riešením je buď „áno" alebo „nie". Napríklad, úlohou



Obrázek 6.3: Triviálny (vľavo) a netriviálny (vpravo) uzol

nie je zistiť, či daná formula je splniteľná, ale chceme nájsť konkrétne priradenie hodnôt premenným, ktoré spĺňa formulu. Podobne v probléme obchodného cestujúceho, nie je ani tak zaujímavé vedieť, či dĺžka najkratšej cesty je menšia ako daná konštatna, ako skôr vedieť, ako vyzerá najkratšia cesta.

Rozhodovacie problémy sú adekvátnymi zástupcami funkcionálnych problémov v kontexte *negatívnych zložitostných výsledkov*. Napríklad, keďže vieme, že problémy CNF a POC(R) sú NP–úplné, tak za predpokladu P $\neq$ NP neexistuje polynomiálny algoritmus na hľadanie spľňajúceho priradenia resp. najkratšej cesty. Obecne ovšem rozhodovací problém môže byť podstatne jednoduchší ako zodpovedajúci funkcionálny. Ukážeme, že v prípade problému splniteľnosti tomu tak nie je.

Príklad 6.21 *FCNF je nasledovný funkcionálny problém: pre danú booleovskú formulu Φ, ak Φ je splniteľná tak úlohou je nájsť nejaké jej spĺňajúce priradenie; inak vrátiť „nie". Pre každý vstup Φ môže mať toto zobrazenie veľa možných hodnôt (každé spĺňajúce priradenie). Nejedná sa teda o skutočnú funkciu asociovanú s rozhodovacím problémom. Avšak pre naše úvahy je to zatiaľ postačujúce. Alternatívne formulácie možno násjť v kapitole o aproximatívnych výpočtoch.*

Zrejme ak problém FCNF je riešiteľný v polynomiálnom čase, tak aj problém CNF je riešiteľný v polynomiálnom čase. Ukážeme, že (prekvapujúco) platí aj opačná implikácia. Ak máme k dispozícii polynomiálny algoritmus na rozhodovanie problému CNF, tak nasledovný polynomiálny algoritmus rieši problém FCNF:

Pre danú formulu Φ algoritmus najprv testuje, či Φ je splniteľná. Ak nie, tak vráti odpoveď „nie". V opačnom prípade hľadá spľňajúce priradenie. Uvažujme formuly $\Phi_{x_1=1}$ a $\Phi_{x_1=0}$, ktoré získame z Φ ak za x_1 dosadíme true *resp.* false*. Evidentne aspoň jedna z nich musí byť splniteľná. Algoritmus rozhodne ktorá (v prípade, že sú obidve, tak zvolí jednu z nich). Zodpovedajúcu hodnotu x_1 substiiuje do Φ a v ďaľšom skúma premennú x_2. Zrejme po nanajvýš $2n$ volaniach predpokladaného polynomiálneho algoritmu, každé s výrazom nie komplikovanejším než Φ, náš algoritmus nájde spĺňajúce priradenie formuly Φ.*

Predchádzajúci algoritmus využíval vlastnosť *self–redukovateľnosti* problému splniteľnosti. Podobný trik je možné použiť aj vprípade iných NP–úplných problémov.

Vzťah medzi rozhodovacími a funkcionálnymi problémami je možné formalizovať. Predpokladajme, že jazyk L patrí do NP. Podľa Lemy 6.1 existuje polynomiálne vyvážená a polynomiálne rozhodnuteľná relácia R_L taká, že pre každý reťazec x platí, že $x \in L$ vtedy a len vtedy, ak existuje y také, že $(x, y) \in R_L$. *Funkcionálny problém asociovaný s jazykom L* je nasledovný výpočtový problém:

> Pre dané x nájdi y také, že $(x, y) \in R_L$; ak žiadne y požadovaných vlastností neexistuje, tak vráť odpoveď „nie".

Triedu všetkých funkcionálnych problémov, asociovaných vyššie uvedeným spôsobom jazykom z triedy NP, nazývame FNP. FP je jej podtrieda obsahujúca len tie problémy, ktoré sú riešiteľné deterministicky v polynomiálnom čase. Napríklad problém FCNF patrí do triedy FNP, ale nie je o ňom známe, či patrí do FP.

Podobne, ako v prípade triedy NP, aj v triede FNP môžeme vyšpecifikovať relatívne najťažšie problémy pomocou pojmu redukcie.

Definícia 6.22 *Funkcionálny problém A je redukovateľný na funkcionálny problém B práve ak je splnené nasledovné: Exisstujú dve funkcie R a S definované na množine reťazcov, obe vypočítateľné v polynomiálnom čase a také, že pre ľubovoľné dva reťazce x a z platí: Ak x je prípadom problému A, tak $R(x)$ je prípadom problému B. Naviac, ak z je správnym riešením prípadu $R(x)$, tak $S(z)$ je správnym riešením prípadu x.*
Funkcionálny problém A je úplný pre triedu FNP*, ak patrí do triedy* FNP *a každý problém z tejto triedy je naň redukovateľný.*

Pretože triedy FP a FNP sú uzavreté vzhľadom na redukovateľnosť, tak FNP–úplné problémy majú analogické vlastnosti ako NP–úplné problémy. Nie je ťažké ukázať, že problém FCNF je úplným pre triedu FNP. Naviac, ako ukazuje príklad 6.21, problém FCNF je riešiteľný v polynomiálnom čase práve ak CNF je riešiteľný v polynomiálnom čase. Z toho už plynie:

Veta 6.23 $\mathsf{FP} = \mathsf{FNP}$ *vtedy a len vtedy ak* $\mathsf{P} = \mathsf{NP}$.

Časť III

Trieda NP

Mnoho problémov, s ktorými sa v praxi môžeme stretnúť je NP–úplných. Vieme, že efektívne (polynomiálne) algoritmy na riešenie týchto problémov môžu existovať, iba ak $\mathsf{P} = \mathsf{NP}$, čo je veľmi nepravdepodobné. To ale neznamená, že tieto problémy sú prakticky neriešiteľné. Skutočnosť, že problém je NP–úplný znamená, že ak $\mathsf{P} \neq \mathsf{NP}$, tak neexistuje polynomiálny algoritmus, ktorý by poskytoval optimálne riešenie pre každý prípad problému. Ak upustíme od tejto striktnej požiadavky, môžeme mnohé problémy vcelku uspokojivo riešiť.

Existuje niekoľko možností, ako možno oslabiť vyššie uvedené požiadavky tak, aby sa problém stal prakticky riešiteľný.

Deterministické algoritmy Pre mnohé z uvažovaných problémov je možné skonštruovať algoritmy, ktorých zložitosť síce nie je polynomiálna, ale superpolynomiálna a sú prakticky použiteľné. Príkladom takéhoto problému je problém batohu. Nevýhodou tohoto prístupu je, že je známych len málo problémov, kde poznáme superpolynomiálny algoritmus; pre väčšinu z nich je najlepší známy algoritmus skutočne exponenciálny.

Iná možnosť je pokúsiť sa nájsť deterministický Turingov stroj $\mathcal{M}$ (algoritmus, program), ktorý rieši problém so zložitosťou napr. $T_{\mathcal{M}}(n) = \frac{1}{100} \cdot 2^{n/50}$. $\mathcal{M}$ má exponenciálnu zložitosť, ale napriek tomu poskytuje uspokojivé riešenie pre vstupy veľkosti až niekoľko sto bitov. Preto ak prípady, ktoré potrebujeme prakticky riešiť, neprekročia danú hranicu, môžeme aj tento (exponenciálny) algoritmus považovať za vyhovujúce riešenie problému.

Ohraničenie problému Často nepotrebujeme riešiť zadaný problém v tak všeobecnom tvare, ako bol pôvodne formulovaný. V mnohých prípadoch môže zúženie problému podstatne znížiť jeho obtiažnosť. Najtypickejšie je ohraničenie množiny prípadov problému Napr. ak (nerozhodnuteľný) Postov korešpondenčný problém uvážime len pre zoznamy nad jednoprvkovou abecedou, tak problém sa stane rozhodnuteľným. Ak sa v probléme CNF obmedzíme na formuly v konjunktívnom normálnom tvare s klauzulami dĺžky nanajvýš 2, máme problém patriaci do P. Podobne ak namiesto Hamiltonovského cyklu v grafe hľadáme Eulerovský cyklus (pokrývajúci všetky hrany).

Pravdepodobnostná analýza vstupu Doteraz sme hodnotili algoritmy podľa ich najhoršieho možného chovania. Pokiaľ dokážeme, že istý algoritmus je vyhovujúci aj v tom najmenej priaznivom prípade, je tento algoritmus z praktického hľadiska veľmi cenný. Ak však naopak ukážeme, že v niektorých prípadoch by trval výpočet neúmerne dlho a bol by prakticky nerealizovateľný, neznamená to, že algoritmus je nepoužiteľný. Je totiž možné, že nepriaznivý vstup sa objaví len zriedkavo, a že v obvyklom prípade prebehne výpočet rýchlo a uspokojivo [4]. Problémom tohoto prístupu je učiť dostatočne presne pravdepodobnostné rozdelenie vstupných hodnôt.

Pravdepodobnostné algoritmy Ďaľšou možnosťou je upustiť od požiadavky, aby algoritmus pre daný prípad problému poskytol *vždy* to isté riešenie, t.j. aby bol deterministický. Obecne je teda možné pokúsiť sa zostrojiť algoritmus, ktorý sa v niektorých krokoch môže náhodne (napr. pomocou generátora náhodných čísel) rozhodnúť medzi niekoľkými možnými pokračovaniami; pritom pripúšťame, že v niektorých prípadoch bude výsledok neuspokojivý, alebo dokonca chybný. Pri hodnotení priemerného chovania sa potom priemer neberie cez všetky možné vstupy istej veľkosti, ale pre všetky možné výpočty určené jedným vstupom. Výsledok analýzy nezávisí na tom, ako často sa jednotlivé vstupy vyskytujú (ako tomu bolo pri pravdepodobnostnej analýze vstupu).

Aproximatívne algoritmy Ak hľadáme algoritmus na riešenie optimalizačného problému, tak môžeme upustiť od požiadavky, aby algoritmus našiel vždy optimálne riešenie. Cieľom je nájsť také algoritmy, ktoré vždy poskytnú riešenie blízke optimálnemu. Pretože v praxi sa často uspokojíme s približným riešením, je tento prísup k riešeniu obtiažnych úloh jeden z najúspešnejších.

[4]Najznámejším príkladom je simplexový algoritmus úlohy lineárneho programovania. Empirická skúsenosť ukazuje, že dĺžka jeho výpočtu je úmerná veľkosti spracovávanej úlohy, zatiaľ čo umelo vykonštruované príklady ukazujú, že táto závislosť môže byť až exponenciálna. Podobné vlastnosti vykazuje triediaci algoritmus Quicksort. Je považovaný za jeden z najrýchlejších triediacich algoritmov napriek tomu, že v najhoršom prípade je čas potrebný na utriedenie n položiek kvadratický.

Pravdepodobnostné aproximatívne algoritmy Kombinácia predchádzajúcich dvoch prístupov vedie k algoritmom, ktoré pre daný prípad optimalizačného problému poskytnú riešenie, ktoré s veľkou pravdepodobnosťou nie je príliš vzdialené od optimálneho riešenia.

Heuristiky Heuristické algoritmy sa podobajú pravdepodobnostným algoritmom v tom, že tiež v priebehu výpočtu robia náhodné rozhodnutia. Rozdiel je v tom, že nie sme schopní dokázať, že heuristický algoritmus poskytuje uspokojivé riešenie problému a to aj napriek tomu, že algoritmus je úspešne používaný na riešenie konkrétneho praktického problému. Príkladom môžu byť genetické algoritmy a algoritmy simulovaného žíhania. Teoretické aj experimentálne výsledky ukazujú, že vyššie uvedené metódy sú spoľahlivejšie než heuristické a že heuristické algoritmy by mali byť používané len v situáciách, keď žiaden z predchádzajúcich prístupov nevedie k vhodnému riešeniu problému.

Poznamenávame ešte, že uvedené prístupy je možné vhodne kombinovať.

Pravdepodobnosté a aproximatívne algoritmy v sebe zahŕňajú nový prístup ku konštrukcii algoritmov a preto sa im budeme venovať podrobnejšie.

Kapitola 7

Pravdepodobnostné výpočty

7.1 Pravdepodobnostné algoritmy

Každý deterministický algoritmus pri viacnásobnom opakovaní výpočtu pre ten istý vstup vypočíta vždy ten istý výstup. Základnou črtou pravdepodobnostného algoritmu je, že pri výpočte sa v ňom „hádže mincou“, a teda pri viacnásobnom opakovaní výpočtu pre ten istý vstup môžeme dostať rôzne výstupy. V pravdepodobnostnom algoritme sa typicky hádže mincou preto, aby sa vybral jeden z možných spôsobov pokračovania vo výpočte. Ukážeme prekvapujúci fakt, že náhodnosť môže priniesť v niektorých prípadoch podstatné zefektívnenie deterministického algoritmu.

Iný pohľad na pravdepodobnostné algoritmy je prostredníctvom nedeterminizmu. Každý nedeterministický krok výpočtu môžeme interpretovať ako náhodné rozhodnutie (hádzanie mincou). Zatiaľ čo pri nedeterministických výpočtoch je existencia jedného prijímajúceho výpočtu postačujúcou podmienkou na akceptovanie vstupu, pri pravdepodobnostných výpočtoch požadujeme "dostatočne veľký" počet akceptujúcich výpočtov. Dôsledkom je, že získaný výsledok je správny s vysokou pravdepodobnosťou. Zvyčajne sa pravdepodobnostný algoritmus opakuje pre ten istý vstup niekoľko krát, čím sa pravdepodobnosť správneho výsledku ešte zvýši.

Príklad 7.1 *Sú dané matice A, B a C rozmerov $n \times n$. Úlohou je zistiť, či $AB \neq C$. Najlepšie známe deterministické riešenie tohoto problému je vynásobiť matice A a B a ich súčin porovnať s maticou C. Časová zložitosť algoritmu je $\mathcal{O}(n^{2.376})$, t.j. zložitosť najlepšieho známeho algoritmu na násobenie matíc. Ukážeme, ako možno tento problém riešiť v čase $\mathcal{O}(n^2)$ využitím pravdepodobnostného prístupu:*

vyber náhodne $x \in \{0,1\}^n$
if $ABx \neq Cx$ **then** *return* $(AB \neq C)$
else *return* $(pravdepodobne AB = C)$

Analýza zložitosti*: Súčin ABx môžeme, využijúc asociativitu, vypočítať ako $A(Bx)$. Súčin matice rozmerov $n \times n$ a n–rozmerného vektora sa dá vypočítať v čase $\mathcal{O}(n^2)$ a teda aj celý algoritmus bude mať zložitosť $\mathcal{O}(n^2)$.*
Odôvodnenie správnosti*:*

1. *Ak $AB = C$, tak $ABx = Cx$ pre všetky x a teda $Pr(ABx = Cx)$=1.*
2. *Ak $AB \neq C$, tak môže platiť rovnosť $ABx = Cx$.*

Ukážeme, že druhý prípad nemôže nastať príliš často.

Lema 7.2 *Predpokladajme, že $AB \neq C$ a x je náhodne vybrané z $\{0,1\}^n$ (uvažujeme uniformné rozdelenie pravdepodobnosti). Potom*

$$Pr(ABx = Cx) \leq \frac{1}{2}.$$

Dôkaz *Z predpokladu $AB \neq C$ plynie $(AB - C) \neq 0$, tj. existujú i, j také, že $(AB - C)_{i,j} \neq 0$. Nech $(a_1, a_2, \ldots, a_n)$ je i–ty riadok matice $AB - C$. Potom platí*

$$\begin{aligned} Pr((AB-C)x = 0) &\leq Pr(\sum_{k=1}^{n} a_k x_k = 0) \\ &= Pr(x_j = \frac{-1}{a_j} \sum_{k \neq j} a_k x_k) \\ &= \frac{1}{2}, \end{aligned}$$

pretože $a_j \neq 0$ a $Pr(x_j = 1) = Pr(x_j = 0) = \frac{1}{2}$. □

Ak algoritmus pre dané matice A, B a C dá odpoveď $AB \neq C$, tak táto odpoveď je určite správna. Avšak ak algoritmus poskytne riešenie $AB = C$, tak existuje možnosť, že odpoveď je nesprávna, t.j. v skutočnosti je $AB \neq C$. Pravdepodobnosť takéhoto chybného výsledku je však nanajvýš $\frac{1}{2}$. Existuje jednoduchý spôsob, ako možno znížiť pravdepodobnosť chybného výsledku: *vykonaním viacerých nezávislých pokusov*. To znamená, že algoritmus opakujeme k–krát, zakaždým pre nový, náhodne vybraný vektor x. Ak pri *každom* opakovaní pokusu dostaneme odpoveď $AB = C$, tak výstupom je $AB = C$; ak aspoň pri jednom opakovaní zistíme $AB \neq C$, tak výstupom je $AB \neq C$. Pravdepodobnosť chybného výstupu sa tak znížila; je nanajvýš $\frac{1}{2^k}$, zatiaľ čo zložitosť algoritmu sa zvýšila na $\mathcal{O}(kn^2)$.

Pravdepodobnostné algoritmy, ktoré majú tú vlastnosť, že jeden typ odpovede je vždy správny (v našom príklade je to odpoveď $AB \neq C$) a druhý typ môže byť chybný ($AB = C$), sa nazývajú algoritmy *typu Monte Carlo*.

Príklad 7.3 *Nech $f(x_1, \ldots, x_n)$ je polynóm n premenných s racionálnymi koeficientami a stupňa k. Chceme rozhodnúť, či f, ako funkcia n premenných, je identicky rovná 0.*

Základnou myšlienkou pravdepodobnostného algoritmu je, že za všetky premenné dosadíme náhodne zvolené čísla a vypočítame hodnotu polynómu. Ak výsledná hodnota je rôzna od 0, tak samozrejme polynóm nemôže byť identicky rovný 0. Avšak môže sa stať, že sme náhodne zvolili práve koreň polynómu a teda výsledná hodnota je 0 aj keď polynóm nie je identicky rovný 0. Dôležité je, že pravdepodobnosť toho, že náhodne zvolíme koreň polynómu, je veľmi malá; t.j. pravdepodobnosť, že urobíme chybu je malá.

Ak by sme ako hodnoty premenných uvažovali reálne čísla, tak pravdepodobnosť chyby by bola 0. Napríklad lineárna rovnica

$$\sum_{i=1}^{n} a_i x_i = a$$

opisuje hyperrovinu; v trojrozmernom priestore kvadratická rovnica

$$x^2 + y^2 - z^2 = 0$$

opisuje povrch kužela. Náhodne vybraný bod bude potom skoro určite ležať mimo tejto plochy. Avšak v realite sa musíme ohraničiť na diskrétne hodnoty premenných; budeme preto predpokladať, že hodnoty premenných vyberáme nezávisle, podľa uniformného rozdelenia, z celočíselného intervalu $[0, \ldots, N-1]$. V takomto prípade bude pravdepodobnosť chyby nenulová a bude závisieť od voľby čísla N; s rastúcim N bude pravdepodobnosť chyby klesať. Presnejšiu formuláciu poskytuje nasledujúca lema.

Lema 7.4 *Nech f je polynóm n premenných a stupňa k, ktorý nie je identicky rovný 0. Ak hodnoty $a_1, \ldots, a_n$ sú vybrané nezávisle a uniformne z intervalu $[0, \ldots, N-1]$, tak*

$$Pr(f(a_1, \ldots, a_n) = 0) \leq \frac{kn}{N}.$$

Dôkaz *Tvrdenie dokážeme matematickou indukciou vzhľadom na počet premenných n. Platnosť tvrdenia pre $n = 1$ vyplýva z faktu, že polynóm jednej premennej stupňa k môže mať nanajvýš k koreňov. Nech $n > 1$; usporiadajme polynóm f podľa mocniny premennej x_1:*

$$f = f_0 + f_1 x_1 + f_2 x_1^2 + \cdots + f_t x_1^t,$$

kde $f_0, \ldots, f_t$ sú polynómy premenných $x_2, \ldots, x_n$; f_t nie je identicky rovný 0 a $t \leq k$. Potom

$$\begin{aligned} & Pr(f(a_1, \ldots, a_n) = 0) \\ \leq\ & Pr(f(a_1, \ldots, a_n) = 0 \mid f_t(a_2, \ldots, a_n) = 0) \cdot Pr(f_t(a_2, \ldots, a_n) = 0) \\ +\ & Pr(f(a_1, \ldots, a_n) = 0 \mid f_t(a_2, \ldots, a_n) \neq 0) \cdot Pr(f_t(a_2, \ldots, a_n) \neq 0) \\ \leq\ & Pr(f_t(a_2, \ldots, a_n) = 0) + Pr(f(a_1, \ldots, a_n) = 0 \mid f_t(a_2, \ldots, a_n) \neq 0). \end{aligned}$$

Prvý člen súčtu môžeme odhadnúť induktívne a druhý je nanajvýš $\frac{k}{N}$ (lebo a_1 je nezávislá od $a_2, \ldots, a_n$; preto ak tieto fixujeme tak, aby $f_t \neq 0$ a teda aj f ako polynóm premennej x_1 bol nenulový, tak pravdepodobnosť, že a_1 je jeho koreňom je nanajvýš $\frac{k}{N}$). Preto

$$Pr(f(a_1, \ldots, a_n) = 0) \leq \frac{k(n-1)}{N} + \frac{k}{N} \leq \frac{kn}{N}.$$

$\square$

Práve dokázaná lema opodstatňuje nasledovný algoritmus na zisťovanie, či daný polynóm f je identicky rovný 0.

vyber náhodne a nezávisle celé čísla $a_1, \ldots, a_n$ z intervalu $[0, \ldots, 2kn]$
<u>**if**</u> $f(a_1, \ldots, a_n) \neq 0$ <u>**then**</u> *return (f nie je identicky rovné 0)*
<u>**else**</u> *return (f pravdepodobne je identicky rovné 0)*

Ak polynóm f je identicky rovný 0, tak navrhnutý algoritmus dá správnu odpoveď. V opačnom prípade je pravdepodobnosť chybnej odpovede nanajvýš $\frac{1}{2}$.

Jednou z mnohých aplikácií uvedeného algoritmu je metóda na zisťovanie, či bipartitný graf má úplné párovanie.

Príklad 7.5 *Je daný bipartitný graf $G = (U, V, E)$, kde $U = \{u_1, \ldots, u_n\}$ a $V = \{v_1, \ldots, v_n\}$ sú množiny jeho vrcholov a $E \subseteq U \times V$. Zaujíma nás, či v grafe G existuje úplné párovanie, t.j. či existuje podmnožina $H \subseteq E$ taká, že každý vrchol grafu inciduje práve s jednou hranou z H. Inými slovami, hľadáme permutáciu π množiny $\{1, \ldots, n\}$ takú, že $(u_i, v_{\pi(i)}) \in H$ pre všetky $u_i \in U$.*

Priraďme každej hrane (u_i, v_j) grafu G symbol x_{ij}. Skonštruujme $n \times n$ bipartitnú incidenčnú maticu M takto:

$$m_{ij} = \begin{cases} x_{ij} & ak\ (u_i, v_j) \in E \\ 0 & inak \end{cases}$$

Pre graf $G = (\{u_1, u_2, u_3\}, \{v_1, v_2, v_3\}, \{(u_1, v_1), (u_1, v_2), (u_2, v_2), (u_2, v_3), (u_3, v_1), (u_3, v_2)\})$ je

$$M = \begin{pmatrix} x_{11} & x_{12} & 0 \\ 0 & x_{22} & x_{23} \\ x_{31} & x_{32} & 0 \end{pmatrix}$$

Determinant $det\, M$ je polynóm stupňa n a premenných x_{ij}, kde každý sčítanec zodpovedá jednému úplnému párovaniu. V našom príklade má graf dve úplné párovania zodpovedajúce dvom členom determinantu

$$det\, M = x_{12}x_{23}x_{31} - x_{11}x_{23}x_{32}$$

Súvislosť medzi existenciou úplneho párovania a determinantom matice možno sformulovať takto:

Lema 7.6 *Bipartitný graf G má úplné párovanie práve vtedy, ak polynóm $det\, M$ nie je identicky rovný 0.*

$$\begin{pmatrix} x_{11} & x_{12} & 0 \\ 0 & x_{22} & x_{23} \\ x_{31} & x_{32} & 0 \end{pmatrix} \Longrightarrow \begin{pmatrix} x_{11} & x_{12} & 0 \\ 0 & x_{22} & x_{23} \\ 0 & x_{32} - \frac{x_{31}}{x_{11}} x_{12} & 0 \end{pmatrix}$$

$$\Longrightarrow \begin{pmatrix} x_{11} & x_{12} & 0 \\ 0 & x_{22} & x_{23} \\ 0 & 0 & -\frac{x_{23}}{x_{22}}(x_{32} - \frac{x_{31}}{x_{11}} x_{12}) \end{pmatrix}$$

Obrázek 7.1: Výpočet determinantu matice M Gaussovou elimináciou

Dôkaz *Pripomeňme si, že*

$$det\, M = \sum_{\pi} \sigma(\pi) \prod_{i=1}^{n} x_{i,\pi(i)},$$

kde π beží cez všetky permutácie množiny $\{1, \ldots, n\}$ a $\sigma(\pi)$ je rovné 1 ak π je súčinom párneho počtu transpozícií; inak je $\sigma(\pi)$ rovné -1. Zrejme člen $m_{1,\pi(1)} \cdot m_{2,\pi(2)} \cdots m_{n,\pi(n)}$ je nenulový, práve ak $(u_i, v_{\pi(i)}) \in E$ pre všetky $i = 1, \ldots, n$, t.j. ak π zodpovedá úplnému párovaniu. Pretože každé dva členy sa líšia aspoň v jednej premennej, nemôžu sa v sume navzájom vyrušiť. Z toho plynie, že G má úplné párovanie práve ak det M nie je identicky rovný 0. □

Každý algoritmus, ktorý vypočíta determinant matice M je teda alternatívnou metódou na riešenie problému úplného párovania. Jednou z obvykle používaných metód je Gaussova eliminácia: maticu upravíme na hormú trojuholníkovú a determinant je rovný súčinu prvkov na diagonále. Ak ale tento postup aplikujeme na našu symbolickú maticu, získame trojuholníkovú maticu, ktorej prvky môžu byť až exponenciálnedlhé výrazy (pozri obrázok 7.5). Spôsob, ako tento problém obísť, spočíva v tom, že za premenné pôvodnej matice dosadíme celé čísla. Dostaneme tak numerickú maticu, ktorej determinant vypočítame Gaussovou elimináciou v polynomiálnom čase. Ak determinant symbolickej matice je nulový, tak aj determinant numerickej matice bude nulový. Ak symbolická matica má nenulový determinant, tak numerická matica môže mať determinant nulový—v prípade, že sme za premenné dosadili korene determinantu symbolickej matice.

Ak vieme, že graf má úplné párovanie, tak ho môžeme nájsť tak, že postupne odstraňujeme hrany grafu a zakaždým zisťujeme, či graf bez práve odstránenej hrany má úplné párovanie. Hoci pre skúmaný problém je známy aj deterministický polynomiálny algoritmus, má navrhnutý polynomiálny Monte Carlo algoritmus tiež svoj význam, pretože rieši obecný problém týkajúci sa determinantov symbolických matíc. Naviac, popísaný algoritmus je možné efektívne paralelizovať.

V predchádzajúcich príkladoch bolo ilustrované použitie pravdepodobnostného prístupu pre problémy, na riešenie ktorých sú známe aj deterministické polynomiálne algoritmy. Nasledujúci polynomiálny Monte Carlo algoritmus rieši problém, pre ktorý nie je známy žiaden deterministický polynomiálny algoritmus.

Príklad 7.7 *Problém prvočíselnosti: pre dané číslo N chceme vedieť, či N je prvočíslo (tzn. jeho jediné delitele sú 1 a N), alebo či N je zložené číslo. Pri návrhu pravdepodobnostného algoritmu vychádzame z nasledovných poznatkov teórie čísel.*

Pre každé celé číslo x, $1 \leq x < N$ označme $W(x)$ nasledujúcu podmienku:

$W(x)$: *buď* $x^{N-1} \not\equiv 1 \; mod\, N$
alebo $\exists$ *celé číslo* M *tvaru* $M = (N-1)\frac{1}{2^i}$ *také, že*
$x^M - 1$ *a* N *majú spoločného deliteľa rôzneho od 1 a N.*

Fakt 7.8 *Ak $W(x)$ platí pre nejaké x, tak N je zložené číslo.*
Ak N je nepárne zložené číslo, tak $W(x)$ platí aspoň pre $(N-1)/2$ čísel x, $1 \leq x < N$.

Uvedený fakt je základom nasledujúceho Milner–Rabinovho pravdepodobnostného tesu, či dané číslo N je zložené.

Milner–Rabinov test
if *N je párne* **then** *return* (*N je zložené číslo*);
náhodne vyber číslo x; $1 < x < N$;
if $W(x)$ **then** *return* (*N je zložené číslo*)
else *return* (*N je pravdepodobne prvočíslo*)

Ak číslo N je prvočíslo, tak algoritmus vždy vráti odpoveď N je prvočíslo. *Ak algoritmus vráti odpoveď* N je zložené, *tak N je skutočne zložené číslo; ak algoritmus vráti odpoveď* N je prvočíslo, *tak s pravdepodobnosťou* $< \frac{1}{2}$ *je táto odpoveď nesprávna, t.j. N je zložené.*

Existuje aj komplementárny pravdepodobnostný polynomiálny algoritmus, ktorý rozpozná zložené čísla s pravdepodobnosťou 1 prvočísla s pravdepodobnosťou aspoň $\frac{1}{2}$.

7.2 Pravdepodobnostné zložitostné triedy

V predchádzajúcej časti sme prezentovali niekoľko konkrétnych pravdepodobnostných algoritmov. Cieľom tejto kapitoly je definovať triedu problémov riešiteľných takýmito algoritmami.

Na prvý pohľad sa zdá, že k tomu, aby sme mohli formálne definovať pravdepodobnostné zložitostné triedy potrebujeme nový výpočtový model—Turingov stroj, ktorý by dokázal „hádzať mincou". Ukážeme, že úplne postačí modifikovať nedeterministický Turingov stroj.

Pre zjednodušenie úvah budeme v ďaľšom pracovať s tzv. štandartizovaným Turingovym strojom. Je definovaný takto:

Definícia 7.9 (Štandartizovaný Turingov stroj) *Štandartizovaný Turingov stroj je nedeterministický stroj* $\mathcal{M}$ *s časovou zložitosťou* $f(n)$, *kde* f *je konštruovateľná funkcia, a s vlastnosťami:*

1. *v každom kroku výpočtu sa* $\mathcal{M}$ *môže rozhodnúť práve medzi dvoma možnosťami, ako ďalej pokračovať vo výpočte,*
2. *každý výpočet stroja* $\mathcal{M}$ *na každom vstupe* w *má dĺžku práve* $f(|w|)$.

Z výsledkov uvedených v kapitole 4 vyplýva, že každý nedeterministický Turingov, ktorého časová zložitosť je konštruovateľná funkcia, sa dá previesť na štandartizovaný stroj.

Definícia 7.10 (Trieda RP) Monte Carlo Turingov stroj $\mathcal{M}$ *je štandartizovaný Turingov stroj s polynomiálnou časovou zložitosťou* $p(n)$ *a taký, že pre každý vstup* w *je buď*

aspoň polovica *spomedzi* $2^{p(|w|)}$ *výpočtov stroja* $\mathcal{M}$ *na* w *je akceptujúcich alebo*

všetky *výpočty* $\mathcal{M}$ *na* w *sú zamietajúce.*

Trieda jazykov rozhodovaných Monte Carlo Turingovymi strojmi sa označuje RP *(skrátené z Randomized Polynomial time).*

Voľne povedané, jazyk L patrí do triedy RP práve ak existuje taký štandartizovaný Turingov stroj $\mathcal{M}$ s polynomiálnou časovou zložitosťou, že

$w \in L$ práve ak *aspoň polovica* spomedzi $2^{p(|w|)}$ výpočtov stroja $\mathcal{M}$ na w je akceptujúcich a

$w \notin L$ práve ak *všetky* výpočty na w sú zamietajúce

Všetky jazyky, ktoré sme uvažovali v predchádzajúcej časti majú Monte Carlo Turingov stroj. Jediným nedeterministickým krokom v týchto algoritmoch bol náhodný výber čísla (čísel) z vopred určeného intervalu. Avšak Turingov stroj môže tento výber previesť tak, že bude najprv hádať prvý bit čísla (t.j. prvý bit binárnej reprezentácie čísla), potom druhý atď. Naopak, kroky v ktorých sa nerobí žiadne nedeterministické rozhodnutie možno ľahko upraviť do tvaru požadovaného pre štandartizovaný stroj: dovolíme, aby sa hádzalo mincou, ale v oboch prípadoch bude výpočet pokračovať rovnako. Ak vstupné slovo nepatrí do jazyka, tak žiaden z výpočtov nemôže skončiť akceptovaním pretože polynomiálny Monte Carlo algoritmus bol konštruovaný ako algoritmus, ktorý nikdy nedá nesprávnu negatívnu odpoveď. Zároveň pravdepodobnosť nesprávnej pozitívnej odpovede je nanajvýš $\frac{1}{2}$ a preto počet zamietajúcich výpočtov Monte Carlo Turingovho stroja na slove patriacom do jazyka je nanajvýš polovičný.

Trieda RP zostane nezmenená, ak konštantu $\frac{1}{2}$ ohraničujúcu pravdepodobnosť nesprávnej pozitívnej odpovede zmeníme. V skutočnosti môžeme namiesto $\frac{1}{2}$ vziať ľubovoľné kladné číslo ostro väčšie ako 0 a menšie než 1. Ak totižto máme algoritmus, v ktorom je pravdepodobnosť nesprávnej pozitívnej odpovede nanajvýš $1-\varepsilon$, kde $\epsilon < \frac{1}{2}$, tak ho môžeme transformovať na taký algoritmus, kde pravdepodobnosť nesprávnej pozitívnej odpovede bude menšia než $\frac{1}{2}$: stačí existujúci algoritmus k–krát zopakovať. Pritom opakovanie v terminológii Turingových strojov znamená, že vo výpočtovom strome na každý list „zavesíme“ strom identický s originálnym. Inak povedané: z každej konfigurácie, v ktorej sa výpočet Turingovho stroja zastavil, sa ešte raz spustí výpočet pôvodného Turingovho stroja. Koncová konfigurácia tohoto nového výpočtu bude akceptujúca, ak aspoň jeden z výpočtov, ktoré do nej viedli bol akceptujúci. Celkove sa tak pravdepodobnosť nesprávnej pozitívnej odpovede zníži a bude menšia než $(1-\varepsilon)^k$. Ak zvolíme konštantu k, t.j. počet opakovaní algoritmu, ako $k = \lceil -\frac{1}{\log(1-\varepsilon)} \rceil$, tak pravdepodobnosť nesprávnej pozitívnej odpovede bude menšia než $\frac{1}{2}$. Zároveň však zložitosť nového algoritmu bude k–násobkom zložitosti originálneho algoritmu a teda polynomiálna. Poznamenávame, že ε môže byť ľubovoľne malá konštanta. Dokonca za ε môžeme vziať aj funkciu tvaru $\frac{1}{p(n)}$, kde $p(n)$ je polynomiálna funckia (počet potrebných opakovaní algoritmu je $k = \lceil -\frac{1}{\log(1-\varepsilon)} \rceil \sim \frac{1}{\varepsilon}$ a teda polynomiálny).

V súvislosti s definovaním novej zložitostnej triedy je prirodzená otázka, aký je jej vzťah k už známym zložitostným triedam. Na každý Monte Carlo Turingov stroj sa môžeme pozerať ako na nedeterministický Turingov stroj s polynomiálnou časovou zložitosťou. Z druhej strany, deterministický Turingov stroj je špeciálnym prípadom Monte Carlo stroja. Celkove teda platí

$$\mathsf{P} \subseteq \mathsf{RP} \subseteq \mathsf{NP}.$$

7.2.1 Trieda ZPP

Doteraz uvažované pravdepodobnostné Monte Carlo algoritmy boli svojim spôsobom asymetrické: na jednej strane existovala nenulová pravdepodobnosť, že odpoveď $w \notin L$ je nesprávna, na druhej strane pozitívna odpoveď bola vždy správna. Je prirodzené predpokladať, že niektoré problémy sa dajú riešiť algoritmami s opačnou vlastnosťou—nenulovou pravdepodobnosťou nesprávnej pozitívnej odpovede a vždy správnou negatívnou odpoveďou. Formálne, trieda co–RP je definovaná ako trieda jazykov, pre ktoré existuje štandartizovaný Turingov stroj polynomiálnej časovej zložitosti a taký, že pre $w \in L$ je každý výpočet stroja akceptujúci a pre $w \notin L$ je aspoň polovica výpočtov stroja je zamietajúcich.

Základnou spoločnou vlastnosťou RP a co–RP algoritmov je, že aj po niekoľkonásobnom opakovaní algoritmu stále existuje nenulová pravdepodobnosť, že získaný výsledok (pozitívny resp. negatívny) je nesprávny. Nikdy nemôžeme s určitosťou povedať, že už sme pokus opakovali dostatočný počet krát na to, aby sme získaný výsledok mohli považovať za definitívne správny.

Rozdielne vlastnosti bude mať ale trieda $\mathsf{RP} \cap \mathsf{co{-}RP}$. Nech na riešenie problému máme k dispozícii dva algoritmy, jeden z nich je typu RP a druhý typu co–RP. Pozitívna odpoveď RP algoritmu je *vždy* správna a negatívna odpoveď co–RP algoritmu je *vždy* správna. Preto ak urobíme niekoľko nezávislých pokusov s *obidvoma algoritmami*, skôr alebo neskôr dostaneme definitívnu odpoveď—buď pozitívnu od prvého algoritmu alebo negatívnu od druhého algoritmu. Po k–násobnom opakovaní oboch algoritmov je pravdepodobnosť toho, že ešte nevieme definitívnu odpoveď menšia ako 2^{-k}. Rozdiel oproti predchádzajúcim algoritmom je v tom, že po určitej dobe budeme poznať definitívnu odpoveď. Nevieme *a priori* (vopred) povedať, koľko nezávislých pokusov musíme vykonať, ale vieme, kedy sme ich už spravili dosť. Algoritmy tohoto typu sa nazývajú *Las Vegas algoritmy*. Trieda jazykov rozhodovaných polynomiálnymi Las Vegas algoritmami sa nazýva ZPP a definuje sa ako

$$\mathsf{ZPP} \stackrel{\text{def}}{=} \mathsf{RP} \cap \mathsf{co–RP}.$$

Podrobnejšie je charakterizovaná nasledujúcou vetou.

Veta 7.11 *Pre ľubovoľný jazyk L sú nasledujúce dve podmienky ekvivalentné:*

(i) $L \in \mathsf{ZPP}$

(ii) Existuje štandartizovaný Turingov stroj $\mathcal{M}$ s polynomiálnou časovou zložitosťou, ktorého každý výpočet skončí v konfigurácii so stavom q_{accept}, q_{reject} alebo q_{stop}. Pritom

- *ak $w \in L$, tak každý výpočet skončí v stave q_{accept} alebo q_{stop}*
- *ak $w \notin L$, tak každý výpočet skončí v stave q_{reject} alebo q_{stop}*
- *pravdepodobnosť, že výpočet skončí v stave q_{stop} je nanajvýš 1/2.*

Dôkaz

(i) ⇒ (ii) Hľadaný stroj $\mathcal{M}$ skonštruujeme takto: pre vstup w simuluje $\mathcal{M}$ výpočet RP stroja na w; ak tento je akceptujúci, tak $\mathcal{M}$ prejde do stavu q_{accept}. V opačnom prípade simuluje výpočet co–RP stroja w; ak tento je zamietajúci, tak $\mathcal{M}$ prejde do stavu q_{reject}, inak prejde do stavu q_{stop}. Pravdepodobnosť že výpočet skončí v stave q_{stop} je nanajvýš $1/2$.

(ii) ⇒ (i) Aby sme ukázali, že $L \in \mathsf{ZPP}$ stačí stroj $\mathcal{M}$ modifikovať tak, že stav q_{stop} nahradíme stavom q_{reject}. Podobne pre triedu co–RP.

□

Polynomiálne algoritmy typu Las Vegas sú z praktického hľadiska veľmi užitočné. Sú spoľahlivé (získaný výsledok je zaručene správny) a výsledok je vypočítaný s veľkou pravdepodobnosťou v polynomiálnom čase. Príkladom problému patriaceho do ZPP je už diskutovaný problém určovania prvočíselnosti. Platí

$$\mathsf{P} \subseteq \mathsf{ZPP} \subseteq \mathsf{NP} \cap \mathsf{co–NP}.$$

7.2.2 Trieda PP

Pri definovaní ďalšieho typu pravdepodobnostných algoritmov upustíme od požiadavky asymetrie, ktorá zaručovala, že pozitívna (resp. negatívna pri co–RP) odpoveď je vždy správna.

Definícia 7.12 (Trieda PP) *Jazyk L patrí do triedy PP práve vtedy, ak existuje polynomiálne časovo ohraničený štandartizovaný Turingov stroj $\mathcal{M}$ taký, že pre každý vstup w,*

$w \in L \Longleftrightarrow$ *viac ako polovica výpočtov stroja $\mathcal{M}$ na vstupe w je akceptujúcich.*

Hovoríme, že jazyk L je „rozhodovaný väčšinou".

Príkladom jazyka patriaceho do triedy PP je jazyk MAJ–CNF (z anglického Majority CNF) obsahujúci také booleovské formuly, pre ktoré viac než polovica spomedzi všetkých možných priradení hodnôt premenným spĺňa danú formulu. Nedeterministický Turingov stroj pre danú booleovskú formulu F n premenných postupne háda hodnoty (0 alebo 1) všetkých n premenných. Výpočet je akceptujúci práve ak uhádnuté priradenie spĺňa formulu F. Zrejme ak F patrí do MAJ–CNF, tak viac než polovica výpočtov na F bude akceptujúcich.

V porovnaní s triedou RP má trieda PP jednu odlišnú vlastnosť: každý nedeterministický Turingov stroj definuje nejaký jazyk z PP. Trieda RP túto vlastnosť nemá; nie pre každý Turingov stroj totižto platí, že ak akceptuje, tak väčšina výpočtov je akceptujúcich. Naviac neexistuje jednoduchý spôsob, ako len z definície (popisu) stroja určiť, či má alebo nemá požadovanú vlastnosť. Jedným z dôsledkov týchto rozdielnych vlastností je, že trieda PP má úplný problém (napr. MAJ–CNF je PP–úplný), ale pre RP nie je známy žiaden.

Opäť nás zaujíma, aký je vzťah triedy PPk ostatným zložitostným triedam.

Lema 7.13

$$\mathsf{NP} \subseteq \mathsf{PP} \subseteq \mathsf{PSPACE}$$

Dôkaz Inklúzia $\mathsf{PP} \subseteq \mathsf{PSPACE}$ vyplýva z toho, že každý nedeterministický polynomiálne časovo ohraničený Turingov stroj sa dá simulovať deterministickým strojom s polynomiálnou priestorovou zložitosťou.

Dokážme platnosť inklúzie $\mathsf{NP} \subseteq \mathsf{PP}$. Predpokladajme, že jazyk $L \in \mathsf{NP}$ je rozhodovaný štandartizovaným strojom $\mathcal{M}$. Skonštruujeme štandartizovaný stroj $\overline{\mathcal{M}}$, rozhodujúci jazyk L väčšinou. $\overline{\mathcal{M}}$ je identický so strojom $\mathcal{M}$ s jedinou výnimkou. $\overline{\mathcal{M}}$ má nový počiatočný stav a z neho dve možnosti nedeterministického výberu. Jedna z možností vedie k simulovaniu stroja $\mathcal{M}$. Druhá možnosť k výpočtom (rovnako dlhým ako sú všetky výpočty stroja $\mathcal{M}$), ktoré sú všetky akceptujúce.

Uvažujme vstup w. Ak $\mathcal{M}$ akceptuje w, tak viac než polovica výpočtov stroja $\overline{\mathcal{M}}$ bude akceptujúcich (aspoň jeden zodpovedajúci akceptujúcemu výpočtu stroja $\mathcal{M}$ plus všetky, ktoré sú akceptujúce vždy). Naopak, ak $\mathcal{M}$ neakceptuje w, tak len polovica spomedzi všetkých výpočtov stroja $\overline{\mathcal{M}}$ je akceptujúcich, čo ale nestačí na akceptovanie slova w strojom $\overline{\mathcal{M}}$. Preto $\overline{\mathcal{M}}$ rozhoduje L väčšinou a $L \in \mathsf{PP}$ □

Trieda PP je uzavretá na komplement; ak zameníme stavy q_{accept} a q_{reject}, dostaneme stroj, ktorý bude rozhodovať komplementárny jazyk s výnimkou slov, pre ktoré je počet akceptujúcich a zamietajúcich výpočtov rovnaký. Tento prípad sa však dá ľahko vyriešiť.

Doteraz sme skúmali tri rôzne pravdepodobnostné zložitostné triedy: RP , ZPP a PP. Avšak zatiaľ čo algoritmy typu RP a ZPP sú realistické, prakticky použiteľné, algoritmy typu PP túto vlastnosť nemajú. Hlavná príčina je v tom, že hranica medzi akceptovaním a zamietnutím je pri týchto algoritmoch veľmi „jemná“. Na to, aby vstupné slovo bolo akceptované stačí, aby počet akceptujúcich výpočtov bol o 2 väčší než počet zamietajúcich výpočtov. Pravdepodobnosť chyby je v tomto (najmenej priaznivom) prípade $1/2 - 1/2^{p(n)}$, kde p je polynóm ohraničujúci časovú zložitosť štandartizovaného stroja. Pri praktickom použití potom nevieme dobre rozlíšiť, o aký vstup sa jedná.

Aby sme predchádzajúce úvahy viac ozrejmili, uvažujme nasledovný príklad. Máme mincu, ktorá nie je vyvážená, to znamená, že častejšie bude padať jedna strana mince než druhá. Vieme, že jedna strana bude padať s pravdepodobnosťou $\frac{1}{2} + \varepsilon$ a druhá s pravdepodobnosťou $\frac{1}{2} - \varepsilon$, pre nejaké $\frac{1}{2} \geq \varepsilon > 0$, ale nevieme, ktorá strana má ktorú pravdepodobnosť. Ako zistíme, ktorá strana padá s väčšou pravdepodobnosťou? Môžeme niekoľkokrát hodiť mincou a stranu, ktorá padne viac krát prehlásiť za stranu s väčšou pravdepodobnosťou. Otázkou je: koľko krát musíme hodiť mincou, aby náš odhad bol s vysokou pravdepodobnosťou správny?

Odpoveď na túto otázku vyplýva zo vzťahu

Fakt 7.14 *Predpokladajme, že $x_1, \ldots, x_N$ sú nezávislé náhodné premenné, ktoré nadobúdajú hodnoty 1 resp. 0 s pravdepodobnosťou p resp. $1-p$. Uvažujme sumu $X = \sum_{i=1}^{N} x_i$. Potom pre všetky $0 \leq \theta \leq 1$ je*

$$\mathtt{Pr}[X \geq (1+\theta)pN] \leq e^{-\frac{1}{3}\theta^2 pN}.$$

Zvoľme $p = \frac{1}{2} - \varepsilon$ a $\theta = \frac{\varepsilon}{\frac{1}{2}-\varepsilon}$; potom pravdepodobnosť, že $\sum_{i=1}^{N} x_i \geq \frac{N}{2}$ je nanajvýš $e^{-\frac{1}{6}\varepsilon^2 N}$. Preto v našom príklade s mincou môžeme dostatočne presne určiť ktorá strana mince padá s pravdepodobnosťou $\frac{1}{2} + \varepsilon$ tak, že vezmeme stranu, ktorá padne častejšie v $N = \frac{1}{\varepsilon^2}$ hádzaniach mincou.

Vráťme sa naspäť k úvahám týkajúcim sa triedy PP. Predpokladajme, že sme štandartizovaný stroj rozhodujúci jazyk $L \in \mathsf{PP}$ spustili N–krát. Nech hodnota náhodnej premennej x_i je 1 práve ak pri i–tom spustení štandartizovaného stroja výpočet na vstupe bol zamietajúci. V najmenej priaznivom prípade však hodnota ε je $2^{-p(n)}$ (p je polynóm ohraničujúci zložitosť stroja) a preto na určenie správnej odpovede s uspokojivou pravdepodobnosťou by sme potrebovali aspoň $\frac{1}{\varepsilon^2} = 2^{2p(n)}$, t.j. exponenciálny, počet opakovaní algoritmu.

Zložitostná trieda, ktorú budeme definovať nasledujúcej časti, je „realistická“ vo vyššie uvedenom zmysle a pritom v porovnaní s triedami RP a ZPP zrejme najobsiahlejšia.

7.2.3 Trieda BPP

Definícia 7.15 (Trieda BPP) *Jazyk L patrí do triedy* BPP *práve vtedy, ak existuje polynomiálne časovo ohraničený štandartizovaný Turingov stroj taký, že pre každý vstup w,*

ak $w \in L$, tak aspoň 3/4 výpočtov stroja na vstupe w je akceptujúcich a

ak $w \notin L$, tak aspoň 3/4 výpočtov stroja na vstupe w je zamietajúcich.

Požadujeme teda, aby stroj prijímal slovo *jasnou väčšinou* a zamietal slovo tiež jasnou väčšinou. S pravdepodobnosťou väčšou ako 3/4 je získaný výsledok správny; pravdepodobnosť chyby je menšia ako 1/4. Trieda BPP zostane nezmenená, ak namiesto čísla 3/4 vezmeme ľubovoľnú konštantu ostro väčšiu než 1/2 a menšiu než 1. Alternatívny pohľad na triedu BPP poskytuje nasledujúca veta.

Veta 7.16 *Jazyk L patrí do triedy* BPP *vtedy a len vtedy, ak pre každý polynóm p sa dá skonštruovať štandartizovaný Turingov stroj rozhodujúci L s polynomiálnou časovou zložitosťou a s pravdepodobnosťou chyby nanajvýš $(\frac{1}{2})^{p(n)}$.*

Dôkaz Implikácia sprava doľava vyplýva z definície.
Pre opačnú implikáciu predpokladajme, že jazyk L patrí do BPP a $\mathcal{M}$ je štandartizovaný stroj akceptujúci L s pravdepodobnosťou chyby $\varepsilon < \frac{1}{2}$. Označme $\delta = (1 - \varepsilon)$ pravdepodobnosť toho, že získaný výsledok je správny. Nech p je ľuboboľný polynóm. Označme $q(n)$ polynóm, spĺňajúci $\frac{1}{2}(2q(n) + 1) > cp(n)$, kde c je konštanta taká, že $(4\varepsilon\delta)^c < \frac{1}{2}$. Skonštruujeme nový stroj $\overline{\mathcal{M}}$ požadovaných vlastností.

Stroj $\overline{\mathcal{M}}$ bude mať dve počítadla. Prvé z nich nastaví na začiatku na 0, druhé na hodnotu $2q(n)+1$, kde n je dĺžka vstupu w. Potom $2q(n)+1$–krát simuluje výpočet stroja $\mathcal{M}$ na vstupe w; zakaždým, keď $\mathcal{M}$ akceptuje, zvýši $\overline{\mathcal{M}}$ hodnotu prvého počítadla o 1. Ak na konci výpočtu je obsah prvého počítadla stroja $\overline{\mathcal{M}}$ väčší ako $q(n)$, tak $\overline{\mathcal{M}}$ akceptuje vstup, inak zamietne vstup.

Nech $m = 2q(n) + 1$. Pravdepodobnosť toho, že práve j fixovaných simulácii stroja $\mathcal{M}$ dá správnu odpoveď je $\delta^j \cdot \varepsilon^{m-j}$. Pravdepodobnosť toho, že práve j simulácií stroja $\mathcal{M}$ (ľubovoľných) dá správnu odpoveď je $\binom{m}{j} \cdot \delta^j \cdot \varepsilon^{m-j}$.

Pravdepodobnosť, že stroj $\overline{\mathcal{M}}$ dá nesprávnu odpoveď je rovná pravdepodobnosti, že nanajvýš $q(n)$ simulácii dá správnu odpoveď, a to je

$$\sum_{j=0}^{q(n)} \binom{m}{j} \cdot \delta^j \cdot \varepsilon^{m-j}.$$

Pretože $\delta > \varepsilon$, je $\delta^j \cdot \varepsilon^{m-j} \leq \delta^{m/2} \cdot \varepsilon^{m/2}$. Pravdepodobnosť chyby stroja $\overline{\mathcal{M}}$ sa teda dá ohraničiť zhora hodnotou

$$\delta^{m/2} \cdot \varepsilon^{m/2} \cdot \sum_{j=0}^{q(n)} \binom{m}{j},$$

ktorá je menšia než $\delta^{m/2} \cdot \varepsilon^{m/2} \cdot 2^m = (4\varepsilon\delta)^{m/2}$. Dosadením za m a za c dostávame, že pravdepodobnosť chyby je menšia ako $(\frac{1}{2})^{p(n)}$. □

Vzťah medzi triedou BPP a ostatnými pravdepodobnostnými zložitostnými triedami je

$$\mathsf{RP} \subseteq \mathsf{BPP} \subseteq \mathsf{PP}.$$

Úplný prehľad definovaných pravdepodobnostných zložitostných tried a vzťahov medzi nimi poskytuje nasledovná schéma ($\longrightarrow$ znamená inklúziu).

$$\begin{tikzcd}
 & & \mathsf{RP} \arrow[r] \arrow[dr] & \mathsf{NP} \arrow[dr] & & \\
\mathsf{P} \arrow[r] & \mathsf{ZPP} \arrow[r] \arrow[ur] \arrow[dr] & \mathsf{NP} \cup \mathsf{co\text{-}NP} \arrow[ur] \arrow[dr] & \mathsf{BPP} \arrow[r] & \mathsf{PP} \arrow[r] & \mathsf{PSPACE} \\
 & & \mathsf{co\text{-}RP} \arrow[r] \arrow[ur] & \mathsf{co\text{-}NP} \arrow[ur] & &
\end{tikzcd}$$

O žiadnej z uvedených inklúzií nie je známe, či je ostrá alebo nie. Pravdepodobnostné algoritmy s ohraničenou pravdepodobnosťou chyby (tzn. tie, pri ktorých môžeme pravdepodobnosť chyby ľubovoľne zmenšiť) môžu byť nesmierne užitočné pri riešení rozhodovacích problémov. Na druhej strane je málo pravdepodobné, že sa podarí nájsť takýto algoritmus na riešenie nejakého NP–úplného problému (implikovalo by to existenciu pravdepodobnostného algoritmu pre ľubovoľný NP–úplný problém!).

Zhrnutím: pravdepodobnostné algoritmy sa používajú na riešenie takých problémov z triedy P, pre ktoré pravdepodobnostné algoritmy sú efektívnejšie než najlepšie známe deterministické algoritmy. Ďalej sa používajú na riešenie problémov, ktorých príslušnosť do triedy P resp. NP–úplnosť je otvorenou otázkou (patrí sem napr. problém určovania prvočíselnosti a problém určiť, či daný polynóm je identicky rovný nule) .

Kapitola 8

Aproximatívne výpočty

Cieľom je nájsť také algoritmy na riešenie optimalizačných problémov, ktoré vždy poskytnú riešenie blízke optimálnemu. Ukazuje sa, že práve tento prístup prináša najviac prakticky použiteľných algoritmov. Cieľom tejto kapitoly je ukázať na niekoľkých príkladoch rôzne typy aproximatívnych algoritmov. Zároveň sa budeme snažiť ukázať, kde sú hranice použiteľnosti tohoto prístupu.

8.1 Aproximatívne algoritmy

Definujeme najprv presne, čo budeme rozumieť pod pojmom optimalizačný problém.

Definícia 8.1 *Optimalizačný problém je charakterizovaný tromi zložkami*

I – *množina vstupných prípadov problému,*

F – *funkcia, ktorá každému vstupnému prípadu $x \in I$ priradí množinu všetkých jeho prípustných riešení,*

c – *cenová funkcia ktorá každému riešeniu $r \in F(x)$ priradí jeho cenu $c(r) \in \mathbb{Q}^+$.*

Maximalizačný problém Π je formulovaný ako úloha pre daný prípad problému násť riešenie s maximálnou cenou. Analogicky je definovaný *minimalizačný problém*, kde úlohou je nájsť riešenie s minimálnou cenou. Optimálnu cenu riešenia označujeme $OPT(x)$; $OPT(x) \stackrel{\text{def}}{=} \max_{r \in F(x)} c(r)$ pre maximalizačný problém a $OPT(x) \stackrel{\text{def}}{=} \min_{r \in F(x)} c(r)$ pre minimalizačný problém.

V ďaľšom budeme pracovať len s optimalizačnými problémami, ktoré *môžu* byť riešiteľné v polynomiálnom čase. Požadujeme preto, aby dĺžka každého prípustného riešenia bola polynomiálne závislá na dĺžke prípadu, aby sme v polynomiálnom čase vedeli overiť, či daný reťazec je prípustným riešením problému, a aby cenová funkcia bola vypočítateľná v polynomiálnom čase. Formálne definujeme triedu NPO obsahujúcu práve tieto problémy.

Definícia 8.2 (Triedy PO a NPO) *Optimalizačný problém (I, F, c) patrí do triedy NPO práve keď sú splnené podmienky:*

1. *Množina I patrí do P.*

2. *Existuje polynóm p taký, že pre každý prípad problému $x \in I$ a pre každé prípustné riešenie $r \in F(x)$ platí $|r| \leq p(|x|)$.*

3. *Pre každé $x \in I$ a každé r také, že $|r| \leq p(|x|)$, je rozhodnuteľné v polynomiálnom čase, či $r \in F(x)$.*

4. *Funckia c je vypočítateľná deterministickým Turingovym strojom v polynomiálnom čase.*

Optimalizačný problém patrí do triedy PO, ak patrí do triedy NPO a je riešiteľný deterministicky v polynomiálnom čase.

Rozhodovacie problémy sú špeciálnym prípadom optimalizačných. Naopak, ku každému optimalizačnému problému môžeme asociovať jazyk (tj. rozhodovací problém) nasledujúcim spôsobom. Namiesto toho, aby sme hľadali optimálne riešenie prípadu x, tak sa budeme pýtať, či existuje riešenie, ktorého cena je aspoň k (resp. nanajvýš k); v prípade kladnej odpovede zaradíme dvojicu $< x, k >$ do jazyka. Inak povedané, jazyk asociovaný maximalizačnému (resp. mimimalizačnému) problému obsahuje všetky také dvojice $< x, k >$, $x \in I$, $k > 0$, pre ktoré existuje $r \in F(x)$ s cenou $c(r) \geq k$ (resp. $c(r) \leq k$). Optimalizačné problémy z triedy NPO, ktorých asociované jazyky sú NP–úplné, sa zvyknú označovať ako NP–*ťažké optimalizačné problémy*.

Vlastnosti definovaných zložitostných tried zhŕňajú nasledovné lemy.

Lema 8.3 *Ak optimalizačný problém patrí do triedy* NPO *resp.* PO, *tak jazyk s ním asociovaný patrí do triedy* NP *resp.* P.

Dôkaz Nech $\Pi =< I, F, c >$ je maximalizačný problém z triedy NPO a nech p je polynóm ohraničujúci dĺžku prípustných riešení. Skonštruujeme nedeterministický Turingov stroj $\mathcal{M}$ rozhodujúci jazyk asociovaný s Π takto:
Stroj $\mathcal{M}$ pre daný vstup $< x, k >$ overí, či x je prípadom problému. Ak áno, tak nedeterministicky uhádne slovo r dĺžky $|r| \leq p(|x|)$. Overí, či $r \in F(x)$. Ak áno, tak vypočíta cenu riešenia $c(r)$; v prípade, že je aspoň k, vstup akceptuje. Polynomiálna zložitosť sroj $\mathcal{M}$ je daná podmienkami 1–4 z definície triedy NPO.

Samozrejme, že ak optimalizačný problém je riešiteľný deterministicky v polynomiálnom čase, tak aj jazyk s ním asociovaný je rozhodnuteľný v polynomiálnom čase. □

Lema 8.4 $\mathsf{P} \neq \mathsf{NP} \implies \mathsf{PO} \neq \mathsf{NPO}$

Dôkaz Existencia deterministického polynomiálneho algoritmu pre ľubovoľný NP–ťažký optimalizačný problém Π implikuje existenciu polynomiálneho algoritmu na riešenie NP–úplného jazyka s ním asociovaného. □

Pre daný NP–ťažký optimalizačný problém Π teda existuje polynomiálny algoritmus, ktorý pre všetky prípady vypočíta optimálne riešenie, iba za predpokladu $\mathsf{P} = \mathsf{NP}$. Ukážeme, ako sa oslabením požiadavky optimality dostaneme k aproximatívnym algoritmom, poskytujúcim „použiteľné“ riešenie.

Definícia 8.5 (Aproximatívny algoritmus) Aproximatívny algoritmus *A pre optimalizačný problém Π je algoritmus polynomiálnej časovej zložitosti, ktorý pre každý prípad x problému Π vypočíta nejaké prípustné riešenie $A(x)$.*

Upozorňujeme, že v našich úvahách sa obmedzujeme len na polynomiálne algoritmy. Samozrejme, riešenie, ktoré nám poskytne aproximatívny algoritmus, môže byť veľmi rozdielne od optimálneho. Preto potrebujeme nejaké kritérium, umožňujúce porovnávať kvalitu získaných riešení. Takýmto kritériom je *chyba* aproximatívneho algoritmu.

8.1.1 Absolútne aproximatívne algoritmy

Definícia 8.6 (Absolútny aproximatívny algoritmus) *Aproximatívny algoritmus A pre optimalizačný problém Π nazveme* absolútny, *ak existuje konšanta $k > 0$ taká, že pre všetky prípady x problému Π je*
$$|c(A(x)) - OPT(x)| \leq k.$$
Číslo k sa nazýva (absolútnou) chybou algoritmu A.

Ukážeme problém, pre ktorý existuje absolútny aproximatívny algoritmus.

PROBLÉM ZAFARBENIE PLANÁRNEHO GRAFU
VSTUP: Neorientovaný planárny graf $G = (V, H)$.
ÚLOHA: Zafarbiť vrcholy grafu najmenším možným počtom farieb tak, aby žiadne dva susedné vrcholy neboli zafarbené tou istou farbou.

Problém rozhodnúť, či graf sa dá zafarbiť 3 farbami je NP-úplný dokonca aj v prípade, že uvažujeme len planárne grafy. Navrhneme absolútny aproximatívny algoritmus pre problém farbenia planárnych grafov.

Vyjdeme z nasledových faktov známych z teórie grafov. Každý planárny graf sa dá zafarbiť 4 farbami a existuje polynomiálny algoritmus, ktorý nájde zafarbenie planárneho grafu 5 farbami.

Aproximatívny algoritmus A pre farbenie planárneho grafu najprv zistí, či graf sa dá zafarbiť nanajvýš dvoma farbami, tj. či je bipartitný (čo je možné v čase úmernom počtu hrán a vrcholov). V opačnom prípade sa aplikuje polynomiálny algoritmus, ktorý nájde zafarbenie piatimi farbami. V žiadnom prípade algoritmus A nepoužije viac než dve nadbytočné farby. Inak povedané, pre každý planárny graf G je

$$c(A(G)) - OPT(G) \leq 2.$$

Predchádzajúci príklad naznačuje, že absolútne aproximatívne algoritmy sa dajú navrhnúť pre také problémy, kde cena optimálneho riešenia sa pohybuje v malom rozpätí a obtiažnosť problému spočíva v jej presnom určení. Absolútny aproximatívny algoritmus využíva tento fakt a poskytuje nejaké tiriviálne riešenie. Otvorenou otázkou je, či aj nejaké skutočne zaujímavé problémy majú absolútne aproximatívne algoritmy.

Negatívne výsledky pre absolútnu aproximáciu

Neexistenciu absolútneho aproximatívneho algoritmu ukážeme pre dva problémy: pre problém BATOH a pre problém KLIKA.

PROBLÉM BATOH
VSTUP: Konečná množina S položiek (objektov), váhová funkcia $v : S \to \mathbb{N}$, hodnotová funkcia $h : S \to \mathbb{N}$, váhový limit $V \in \mathbb{N}$. Prípustným riešením daného prípadu je podmnožina $S' \subseteq S$ taká, že $\sum_{a\in S'} v(a) \leq V$. Cenou riešenia S' je $c(S') = \sum_{a\in S'} h(a)$.
ÚLOHA: Nájsť riešenie s maximálnou cenou.

Neformálne: do batohu fixovanej váhovej kapacity V chceme uložiť položky rôznej váhy tak, aby sme pritom maximalizovali hodnotu uložených vecí. Jazyk asociovaný s týmto problémom je práve jazyk H–BATOH, o ktorom sme v leme 5.16 dokázali, že je NP–úplný.

Veta 8.7 *Ak* $\mathsf{P} \neq \mathsf{NP}$, *tak pre problém BATOH neexistuje žiaden absolútny aproximatívny algoritmus.*

Dôkaz Dôkaz budeme viesť sporom. Predpokladajme, že pre problíém BATOH existuje algoritmus A, ktorého absolútna chyba je k, kde k je nejaké prirodzené číslo. Využijúc A navrheme nový (polynomiálny) algoritmus, ktorý pre ľubovoľný prípad problému BATOH nájde jeho optimálne riešenie. Aplikáciou lemy 8.4 dostávame P= NP, čo je spor s predpokladom.

Hľadaný polynomiálny algoritmus pre daný prípad x problému BATOH najprv skonštruuje nový prípad x' a potom simuluje výpočet algoritmu A na vstupe x'. Prípad x' dostaneme ak položíme $v' = v$ a $h'(a) = (k+1)h(a)$. Inak povedané, nový prípad x' sa od pôvodného líši len nastavením hodnôt položiek, ktoré sa zväčšili $(k+1)$ násobne. Evidentne majú oba prípady tie isté prípustné riešenia, $F(x) = F(x')$. Jediný rozdiel je v tom, že cena prípustných riešení problému x' je $(k+1)$ násobkom cien riešení prípadu x.

Algoritmus A na vstupe x' vypočíta riešenie $A(x')$. Pretože absolútna chyba algoritmu A je k, tak

$$|\sum_{a\in A(x')} h'(a) - OPT(x')| \leq k.$$

Vzhľadom na nastavenie hodnotovej funkcie prípadu x' ke $OPT(x') = (k+1)OPT(x)$. Preto platí

$$|\sum_{a\in A(x')} (k+1)h(a) - (k+1)OPT(x)| \leq k.$$

Úpravou dostávame

$$|\sum_{a\in A(x')} h(a) - OPT(x)| \leq \frac{k}{k+1}.$$

Pretože pracujeme s celými číslami, je

$$|\sum_{a\in A(x')} h(a) - OPT(x)| = 0$$

a teda $A(x')$ je optimálnym riešením prípadu x. □

Uvedená konštrukcia je založená na tom, že, vďaka lineárnej závislosti cenovej funkcie od vstupu, sme mohli daný prípad potrebným spôsobom „naškálovať". Iný pohľad na problém absolútnej aproximácie poskytuje problém KLIKA.

PROBLÉM KLIKA
VSTUP: Orientovaný graf G. Prípustným riešením prípadu G je každý úplný podgraf grafu G. Cenou riešenia je veľkosť podgrafu, tj. počet jeho vrcholov.
ÚLOHA: Nájsť riešenie s maximálnou cenou.

V kapitole 5 sme ukázali, že príslušný rozhodovací problém k–KLIKA je NP–úplný.

Veta 8.8 *Ak* $\mathsf{P} \neq \mathsf{NP}$*, tak pre problém KLIKA neexistuje žiaden absolútny aproximatívny algoritmus.*

Dôkaz Definujeme najprv pojem m–tej mocniny grafu G, značíme G^m. Vezmeme m kópií grafu G a hranou spojíme každé dva vrcholy, ktoré neležia v tej istej kópii. Ľahko overíme, že ak najväčší úplný podgraf grafu G má veľkosť α, tak najväčší úplný podgraf grafu G^m má veľkosť $m\alpha$.

Predpokladajme opäť, že pre problém KLIKA existuje aproximatívny algoritmus A, ktorého absolútna chyba je k. Potom optimálne riešenie problému KLIKA možno nájsť tak, že spustíme A na vstupe G^{k+1}. Platí

$$|c(A(G^{k+1})) - OPT(G^{k+1})| \leq k.$$

Ľahko vidno, že ak máme kliku veľkosti β grafu G^m, tak kliku veľkosti $\lceil \frac{\beta}{m} \rceil$ grafu G dokážeme nájsť v polynomiálnom čase. To ale značí, že vieme nájsť kliku K v G takú, že

$$|c(K) - OPT(G)| \leq \frac{k}{k+1}.$$

Hodnoty $c(K)$ aj $OPT(G)$ sú celočíselné a preto K musí byť optimálna klika. □

8.1.2 Relatívne aproximatívne algoritmy

Definícia 8.9 (ϵ–aproximatívny algoritmus) *Aproximatívny algoritmus A pre maximalizačný problém Π nazvemem* ϵ–aproximatívny *(ϵ je konštanta, $1 > \epsilon \geq 0$), ak pre všetky prípady x problému Π platí*

$$\frac{OPT(x) - c(A(x))}{OPT(x)} \leq \epsilon.$$

V prípade, že Π je minimalizačný problém, požadujeme aby

$$\frac{c(A(x)) - OPT(x)}{c(A(x))} \leq \epsilon.$$

Alternatívne možno vlastnosť ϵ–aproximatívneho algoritmu vyjadriť ako vzdialenosť poskytnutého riešenia od optimálneho riešenia. ϵ–aproximatívny algoritmus poskytne pre každý prípad maximalizačného problému riešenie, ktoré nie je menšie ako $(1-\epsilon)$ násobok optimálneho riešenia, tj.

$$c(A(x)) \geq (1-\epsilon)OPT(x) \tag{8.1}$$

Pre minimalizačný problém získané riešenie nikdy nie je väčšie ako $\frac{1}{1-\epsilon}$ násobok optima,tj.

$$c(A(x)) \leq \frac{1}{1-\epsilon}OPT(x) \tag{8.2}$$

Číslo ϵ sa nazýva *relatívna chyba* algoritmu. Platí, že čím menšia je hodnota ϵ, tým kvalitnejšie (bližšie optimu) riešenie algoritmus poskytuje. Preto pre každý optimalizačný problém Π nás zaujíma, pre aké najmenšie ϵ existuje polynomiálny ϵ–aproximatívny algoritmus pre problém Π. V niektorých prípadoch žiadne takéto najmenšie ϵ neexistuje; je možné konštruovať ľubovoľne presné algoritmy.

Definícia 8.10 (Aproximatívny prah) Aproximatívny prah *optimalizačného problému* Π *je maximálne číslo* p *také, že pre všetky* $\epsilon \geq p$ *existuje* ϵ*–aproximatívny algoritmus pre problém* Π.

Uvedieme niekoľko príkladov problémov, pre ktoré existujú relatívne aproximatívne algoritmy.

PROBLÉM VRCHOLOVÉ POKRYTIE
VSTUP: Neorientovaný graf $G = (V, H)$. Prípustným riešením je každá množina $M \subseteq V$ taká, že pre každú hranu $(i, j) \in H$ aspoň jeden z vrcholov i, j patrí do M. Cenou riešenia je kardinalita množiny M.
ÚLOHA: Nájsť riešenie s minimálnou cenou.

Lema 5.13 obsahuje dôkaz, že jazyk asociovaný s problémom VRCHOLOVÉ POKRYTIE je NP–úplný. Ukážeme, ako preň možno skonštruovať aproximatívny algoritmus.

Prirodzená úvaha, z ktorej by sme mohli pri konštrukcii algoritmu vychádzať je takáto: Ktorý vrchol je výhodné zaradiť do množiny M? Zrejme vrchol s čo najväčším stupňom, pretože ten pokrýva súčasne veľa hrán. Zodpovedajúci algoritmus by bol:

VSTUP: graf $G = (V, H)$
$M := \emptyset$
while $H \neq \emptyset$ **do** *nájdi vrchol* v *grafu* G *s najväčším stupňom*
    $M := M \cup \{v\}$
    $V := V \setminus \{v\}$
    $H := H \setminus \{\, (u, v) \mid (u, v) \in H \}$
od

Tento algoritmus však nie je ϵ–aproximatívny, pretože relatívna chyba môže rásť spolu s počtom vrcholov grafu (napríklad pre binárne stromy) a preto ju nemožno ohraničiť žiadnou konštantou menšou ako 1. Na zostrojenie ϵ–aproximatívneho algoritmu pre problém VRCHOLOVÉHO POKRYTIA musíme použiť jemnejšiu metódu:

VSTUP: graf $G = (V, H)$
$M := \emptyset$
while $H \neq \emptyset$ **do** *vyber ľubovoľnú hranu* $(u, v) \in H$
    $M := M \cup \{u, v\}$
    $V := V \setminus \{u, v\}$
    $H := H \setminus \{\, (u, u'), (v, v') \mid (u, u'), (v, v') \in H \}$
od

Výstupom navrhnutého algoritmu je množina M. Aký je jej vzťah k optimálnemu pokrytiu? Pri vytváraní množiny M sme z grafu G odstránili $\frac{1}{2}|M|$ hrán, z ktorých žiadne dve nemali spoločný vrchol. Každé pokrytie, vrátane optimálneho, musí obsahovať aspoň jeden vrchol každej z týchto hrán (inak by hrana nebola pokrytá). Preto $OPT(G) \geq \frac{1}{2}|M|$, a teda

$$\frac{|M| - OPT(G)}{|M|} \leq \frac{1}{2}.$$

Fakt 8.11 *Aproximatívny prah problému VRCHOLOVÉ POKRYTIE je nanajvýš* $\frac{1}{2}$.

Ďalší problém, pre ktorý je známy ϵ–aproximatívny algoritmus, je rozvrhovací problém (Multiprocessor Scheduling).

PROBLÉM ROZVRHOVANIE
VSTUP: Je daných n úloh $U_1, \ldots, U_n$. Každej úlohe U_i je priradený čas p_i potrebný na jej vykonanie. Okrem toho je daných m identických strojov (procesorov). Prípustným riešením je

každé rozdelenie n úloh na m podmnožín, predstavujúce priradenie úloh procesorom. Cenou riešenia je celkový čas potrebný na vykonanie všetkých n úloh. Celkový čas sa definuje ako maximum cez súčet časov potrebných na vykonanie úloh priradených jednotlivým procesorom.
ÚLOHA: Nájsť riešenie s minimálnou cenou.

Jazyk asociovaný s týmto problémom je NP–ťažký dokonca aj v prípade, že $m = 2$. Uvažujme preň nasledujúci rozvrhovací algoritmus A. Algoritmus berie úlohy jednu po druhej a každej priradí jeden z m procesorov. Priraďovanie sa riadi nasledovným pravidlom: úlohe sa priradí procesor, ktorý je v danom okamihu najmenej obsadený. Obsadením procesoru rozumieme súčet časových zložitostí úloh, ktoré mu už boli priradené.

Lema 8.12 *Nech A je vyššie uvedený rozvrhovací algoritmus. Potom pre každý prípad x rozvrhovacieho problému je*
$$\frac{c(A(x)) - OPT(x)}{c(A(x))} \leq \frac{m-1}{2m-1}.$$
Navyše, táto hranica je tesná, t.j. existuje vstup x^* pre ktorý je
$$\frac{c(A(x^*)) - OPT(x^*)}{c(A(x^*))} = \frac{m-1}{2m-1}.$$

Dôkaz Ukážeme, že pre ľubovoľný prípad x je $c(A(x)) \leq (2 - \frac{1}{m})OPT(x)$. Z toho už ľahko na základe vzťahu 8.2 odvodíme, že relatívna chyba algoritmu A je $\frac{m-1}{2m-1}$.

Predpokladajme, bez straty všeobecnosti, že po rozvrhnutí všetkých úloh je procesor P_1 najviac obsadený; jeho obsadenie je práve $c(A(x))$. Ďalej nech U_j je posledná úloha, ktorá mu bola priradená.

Tvrdíme, že obsadenie každého procesora je aspoň $c(A(x)) - p_j$. V okamihu, keď úloha U_j bola priradená procesoru P_1, bol tento procesor najmenej obsadený; jeho obsadenie bolo práve $c(A(x)) - p_j$. Preto
$$\sum_{i=1}^{n} p_i \geq m(c(A(x)) - p_j) + p_j$$
Zároveň ale platí
$$OPT(x) \geq \frac{\sum_{i=1}^{n} p_i}{m}$$
pretože na konci rozvrhovacieho algoritmu musí mať aspoň jeden procesor takéto obsadenie. Zhrnutím dostávame
$$OPT(x) \geq (c(A(x)) - p_j) + \frac{p_j}{m}$$
$$OPT(x) + p_j\frac{m-1}{m} \geq c(A(x))$$
Ak teraz vezmeme do úvahy, že $p_j \leq OPT(x)$ (niektorý proces musí previesť úlohu p_j), tak dostaneme potrebný odhad relatívnej chyby.

Aby sme ukázali, že získaný odhad je tesný, uvažujme nasledujúci prípad x^* rozvrhovacieho problému. Nech $n = m(m-1) + 1$ a nech na vykonanie prvých $n - 1$ úloh je potrebný čas 1, zatiaľ čo na vykonanie poslednej úlohy je potrebný čas m. Ľahko vidno, že $OPT(x^*) = m$, ale $A(x^*) = 2m - 1$. □

Uvedený algoritmus nie je najlepší známy algoritmus na riešenie rozvrhovacieho problému. Napríklad, ak v už uvedenom algoritme zaradíme pred samotný výpočet procedúru, ktorá najprv zoradí všetky úlohy podľa času potrebného na ich vykonanie v zostupnom poradí, tak relatívna chyba algoritmu sa zníži na $\frac{m-1}{4m-1}$.

Ako posledný uvádzame ϵ–aproximatívny algoritmus pre problém hľadania maximálneho rezu v grafe.

PROBLÉM MAXIMÁLNY REZ
VSTUP: Je daný graf $G = (V, H)$. Prípustným riešením je rozdelenie množiny vrcholov na dve

podmnožiny S a $V - S$. Cenou riešenia je počet hrán spájajúcich vrcholy z S a z $V - S$.
ÚLOHA: Nájsť riešenie s maximálnou cenou.

Jazyk asociovaný s problémom nájdeniať maximálneho rezu v grafe je NP–úplný. Nami prezentovaný aproximatívny algoritmus je založený na myšlienke tzv. lokálneho vylepšenia. Začneme s ľubovoľným rozdelením množiny vrcholov grafu $G = (V, H)$ (dokonca môže byť aj $S = \emptyset$) a opakujeme nasledujúci krok: Ak pridaním (alebo odobraním) jediného vrcholu z množiny S možno rez zväčšiť (bude obsahovať viac hrán), tak tento vrchol do S pridáme (alebo odoberieme). Ak žiadne zväčšenie nie je možné, výpočet algoritmu končí.

Konečnosť algoritmu je zaručená, lebo v každom kroku sa veľkosť rezu zvýši aspoň o 1 a jeho veľkosť pritom nemôže presiahnuť počet hrán grafu. Naviac tvrdíme, že veľkosť rezu, ktorý takto získame, bude aspoň polovičná v porovnaní s optimálnym rezom.

Uvažujme rozklad množimy vrcholov na štyri disjunktné podmnožiny $V = V_1 \cup V_2 \cup V_3 \cup V_4$ také, že rozdelenie získané našim algoritmom je práve $(V_1 \cup V_2, V_3 \cup V_4)$, zatiaľ čo optimálne rozdelenie množimy vrcholov je $(V_1 \cup V_3, V_2 \cup V_4)$. Nech h_{ij}, pre $1 \leq i, j \leq 4$, je počet hrán medzi vrcholmi z V_i a z V_j. Všetko, čo vieme o našom rozdelení je, že ho nemožno zlepšiť presunutím žiadneho vrchola z jednej množiny do druhej. Takže pre každý vrchol z V_1 počet hrán vedúcich z tohoto vrchola do vrcholov z V_3 a z V_4 prevyšuje počet hrán, ktoré vedú z tohoto vrchola do vrcholov z V_1 a z V_2. Ak vezmeme všetky vrcholy z V_1, dostávame nerovnosť $2h_{11} + h_{12} \leq h_{13} + h_{14}$, z čoho plynie $h_{12} \leq h_{13} + h_{14}$. Analogicky pre ostatné tri množiny vrcholov dostaneme:

$$\begin{aligned} h_{12} &\leq h_{23} + h_{24} \\ h_{34} &\leq h_{23} + h_{13} \\ h_{34} &\leq h_{14} + h_{24} \end{aligned}$$

Aritmetickou úpravou z uvedených štyroch nerovností získame

$$h_{12} + h_{34} + h_{14} + h_{23} \leq 2 \cdot (h_{13} + h_{14} + h_{23} + h_{24}),$$

čo je vlastne tvrdenie, že získané riešenie je aspoň polovička optimálneho.

Fakt 8.13 *Aproximatívny prah problému MAXIMÁLNY REZ je nanajvýš* $\frac{1}{2}$.

Algoritmus založený na technike lokálneho vylepšenie je možné navrhnúť pre mnoho optimalizačných problémov. Nevýhodou je, že len zriedkakedy vieme pre takúto heuristiku presne analyzovať jej časovú zložitosť a odhadnúť veľkosť relatívnej chyby.

8.2 Aproximatívne schémy

Ďalšiu triedu optimalizačných problémov tvoria problémy, pre ktoré je možné skonštruovať aproximatívne algoritmy s ľubovoľne malou relatívnou chybou. V tejto časti sa pokúsime charakterizovať problémy tohoto typu. Najprv spresníme tvrdenie „je možné skonštruovať aproximatívne algoritmy s ľubovoľne malou relatívnou chybou".

Definícia 8.14 (Polynomiálna aproximatívna schéma) *Nech* Π *je optimalizačný problém,* x *prípad problému* Π *a* ϵ *racionálne číslo, 1* $> \epsilon \geq 0$. Polynomiálna aproximatívna schéma *pre problém* Π *je algoritmus A, ktorý pre vstup* $< x, \epsilon >$ *vypočíta, v čase polynomiálnom vzhľadom k dĺžke* x*, prípustné riešenie prípadu* x*, ktorého relatívna chyba je nanajvýš* ϵ.

Upozorňujeme, že na algoritmus A sa môžeme pozerať ako na triedu algoritmov $\{A_\epsilon \mid \epsilon > 0\}$, kde relatívna chyba algoritmu A_ϵ je ϵ. V definícii sa naviac požaduje, aby táto (nekonečná) trieda algoritmov mala konečnú a uniformnú reprezentáciu (tj. bola rekurzívna).

Vráťme sa k problému ROZVRHOVANIE, ktorý sme skúmali v predchádzajúcej časti. Ukážeme, že optimalizáciou tam navrhnutého algoritmu možeme získať aproximatívnu schému.

Predpokladajme, že $n > m$ (počet úloh je väčší ako počet procesorov) a že úlohy sú zoradené podľa času potrebného na ich vykonanie v nerastúcom poradí, tj. $i < j$ implikuje $p_i \geq p_j$ (to môžeme docieliť aplikáciou triediaceho algoritmu). Uvažujme teraz algoritmus A_k, ktorý je definovaný pre každé celé číslo z intervalu $1, \ldots, n$.

Algoritmus A_k

1. krok Rozvrhni optimálne prvých k úloh $U_1, \dots, U_k$.

2. krok Štartujúc s rozvrhnutím získaným v 1. kroku, rozvrhni ostatné úlohy takto: úlohy sa berú jedna po druhej a každej sa priradí jeden z m procesorov. Priraďovanie sa riadi nasledovným pravidlom: úlohe sa priradí procesor, ktorý je v danom okamihu najmenej obsadený. Obsadením procesoru rozumieme súčet časových zložitostí úloh, ktoré mu už boli priradené.

Lema 8.15 *Relatívna chyba algoritmu* A_k *je*

$$1 - \frac{1}{1 + \frac{m-1}{m\lceil\frac{k}{m}\rceil}}.$$

Dôkaz Nech K označuje čas rozvrhnutia nájdeného v prvom kroku algoritmu A_k pre vstup x. Zrejme, ak $c(A_k(x)) = K$, tak algoritmus našiel optimálne rozvrhnutie a tvrdenie je dokázané. Predpokladajme, že čas rozvrhnutia nájdeného algoritmom A_k je ostro väčší ako K. Potom musí existovať úloha U_j, pre nejaké $j > k$, taká, že jej vykonanie skončilo v čase $c(A_k(x))$. To implikuje, že v časovom intervale $[0, c(A_k(x)) - p_j]$ boli všetky procesory obsadené, pretože inak by úloha U_j bola vykonávaná už skôr. Z toho plynie

$$\sum_{i=1}^{n} p_i \quad \geq \quad m(c(A_k(x)) - p_j) + p_j$$

Keďže úlohy boli zoradené podľa času potrebného na ich vykonanie, je $p_j \leq p_{k+1}$ a dosadením

$$\sum_{i=1}^{n} p_i \quad \geq \quad mc(A_k(x)) - (m-1)p_{k+1}$$

Ak teraz využijeme, že $OPT(x) \geq \frac{\sum_{i=1}^{n} p_i}{m}$, tak máme

$$c(A_k(x)) \quad \leq \quad OPT(x) + (1 - \frac{1}{m})p_{k+1}$$

Ak sa nám teraz podarí ukázať, že p_{k+1} nie je v porovnaní s $OPT(x)$ príliš veľké, dostaneme hľadaný odhad relatívnej chyby [1]. To sa dá urobiť takto: Uvažujme k najzložitejších úloh, ktoré boli rozvrhnuté v prvom kroku algoritmu. V optimálnom rozvrhnutí musí byť niektorému procesoru priradených aspoň $\lceil\frac{k}{m}\rceil$ z týchto k úloh. Pritom čas potrebný na vykonanie každej z nich je aspoň p_{k+1}. Zhrnutím

$$\begin{aligned} OPT(x) \quad &\geq \quad \left\lceil\frac{k}{m}\right\rceil p_{k+1} \\ p_{k+1} \quad &\leq \quad \frac{OPT(x)}{\lceil\frac{k}{m}\rceil} \end{aligned}$$

Dosadením do predošlej nerovnosti dostávame

$$c(A_k(x)) \leq OPT(x) \cdot (1 + \frac{m-1}{m\lceil\frac{k}{m}\rceil}).$$

Z toho už plynie hľadaný odhad relatívnej chyby. □

Teraz už sme pripravení navrhnúť polynomiálnu aproximatívnu schému pre problém ROZVRHOVANIE. Nech A_ϵ, pre ľubovoľné $\epsilon > 0$ je algoritmus A_k s k vybraným tak, aby relatívna

[1] Odhad relatívnej chyby pozostáva väčšinou z dvoch častí. Najprv sa nájde horná hranica pre $c(A_k(x))$, často vyjadrená pomocou $OPT(x)$ a nejakého parametra R. Potom sa nájde dolná hranica pre $OPT(x)$, vyjadrená aj s pomocou $c(A_k(x))$ a R. Eliminovaním R z týchto dvoch nerovností už dostaneme hľadanú nerovnosť.

chyba algorimtu A_k bola nanajvýš ϵ. Ľahko sa overí, že stačí vziať $k \geq \frac{1-\epsilon}{\epsilon}m$. V popise algoritmu A_k sme ešte nešpecifikovali krok 1. Na jeho realizáciu stačí použiť algoritmus úplného prehľadávania, ktorý nájde optimálne rozvrhnutie pre k úloh na m procesoroch v čase $\mathcal{O}(m^k)$. Časová zložitosť kroku 1 je teda polynomiálna vzhľadom na dĺžku vstupu pre dostatočne malé hodnoty m, povedzme pre m konštantné. Preto platí nasledovná veta.

Veta 8.16 *Pre fixované m, existuje polynomiálna aproximatívna schéma pre problém rozvrhovania pre m procesorov.*

Poznamenávame, že tento algoritmus má malý praktický význam dokonca aj pre malé hodnoty m. Časová zložitosť prvého kroku je $\mathcal{O}(m^k)$ a teda exponenciálna v $\frac{1}{\epsilon}$. Pre relatívnu chybu blížiacu sa k nule je prakticky nepoužiteľný. Uvedený trade-off medzi časovou zložitosťou a kvalitou aproximácie je dôležitou charakteristikou každej aproximatívnej schémy. Dobré sú také schémy, kde so znižovaním relatívnej chyby nerastie časová zložitosť príliš rýchlo. Túto vlastnosť majú úplné aproximatívne schémy.

Definícia 8.17 (Úplná aproximatívna schéma) *Polynomiálna aproximatívna schéma sa nazýva* úplná*, ak jej časová zložitosť je polynomiálna súčasne vzhľadom k dĺžke prípadu* a *k* $\frac{1}{\epsilon}$.

Problémom, pre ktorý je známa úplná aproximatívna schéma je problém BATOH. Predpokladajme, že máme daný prípad x problému BATOH. To znamená, že máme množinu S obsahujúcu n položiek; položka a má váhu $v(a)$ a hodnotu $h(a)$; naviac máme daný váhový limit V. Úlohou je nájsť podmnožinu $S' \subseteq S$ takú, že $\sum_{a\in S'} v(a) \leq V$ a pritom suma $\sum_{a\in S'} h(a)$ je maximálna.

Pri konštrukcii úplnej aproximatívnej schémy vyjdeme z pseudopolynomiálneho algoritmu pre problém BATOH popísaného v leme 5.17. Jeho jednoduchou modifikáciou dostaneme algoritmus, ktorý vypočíta nielen cenu, ale aj samotné optimálne riešenie.

Zložitosť skúmaného algoritmu je $\mathcal{O}(n \cdot V)$ a je úmerná váham jednotlivých položiek. Pri navrhovaní úplnej aproximatívnej schémy využijeme duálny prístup; namiesto s váhami budeme pracovať s hodnotami.

Nech $H = \max_{a\in S}\{h(a)\}$ je maximálna hodnota; pre každé $i = 0, 1, \ldots, n$ a $w = 0, 1, \ldots, nH$ definujeme číslo $W(i, w)$ ako minimálnu váhu, ktorú môžeme dosiahnuť výberom spomedzi prvých i položiek tak, aby ich celková hodnota bola presne w. Položíme $W(0, w) = \infty$ pre všetky w a ďalej počítame

$$W(i+1, w) = \min\{W(i, w), (W(i, w - h(a_{i+1})) + v(a_{i+1}))\}.$$

Tu a_{i+1} označuje $i+1$–vý prvok množiny S. Na konci vyberieme najväčšie w také, že $W(n, w) \leq V$. Počet krokov potrebných na nájdenie optimálneho riešenia je tak $\mathcal{O}(n^2 H)$. Keď však uvážime, že na zakódovanie prípadu x problému BATOH potrebujeme $\mathcal{O}(n(\log H + \log V))$ bitov ($V = \max_{a\in S}\{v(a)\}$) dostávame, že sa jedná o pseudo–polynomiálny a nie polynomiálny algoritmus. Avšak my sa uspokojíme s približným riešením: preto si môžeme dovoliť zanedbať niekoľko najmenej významných bitov čísel, s ktorými narábame. Docielime tak zvýšenie rýchlosti algoritmu na úkor jeho presnosti.

Pre daný prípad x definujeme aproximatívny prípad x', ktorý sa od pôvodného líši len nastavením hodnôt položiek: nová hodnota položky a je $h'(a) = 2^b \lfloor \frac{h(a)}{2^b} \rfloor$; t.j. staré hodnoty, v ktorých je b najmenej významných bitov nahradených nulami; b je parameter, ktorý špecifikujeme neskôr.

Ak teraz namiesto prípadu x budeme riešiť aproximatívny prípad x', tak počet krokov potrebných na výpočet tabuľky bude $\mathcal{O}(\frac{n^2 H}{2^b})$, pretože posledné nuly v hodnotách položiek môžeme ignorovať. Nasledujúce vzťahy ukazujú, ako sa líši cena optimálneho riešenia R' aproximatívneho prípadu x' od ceny optimálneho riešenia R prípadu x:

$$\sum_{a\in R} h(a) \geq \sum_{a\in R'} h(a) \geq \sum_{a\in R'} h'(a) \geq \sum_{a\in R} h'(a) \geq \sum_{a\in R} (h(a) - 2^b) \geq \sum_{a\in R} h(a) \ - \ n2^b.$$

Platnosť prvej nerovnosti vyplýva z optimality riešenia R prípadu x; druhá nerovnosť je daná vzťahom $h'(a) \leq h(a)$; tretia optimalitou riešenia R' prípadu x'; nasledujúca plynie z

$h'(a) \geq h(a) - 2^b$ a posledná platí preto, že $|R| \leq n$. Porovnaním druhého a posledného výrazu vidíme, že cena riešenia získaného našim aproximatívnym algoritmom je nanajvýš o $n2^b$ menšia, než cena optimálneho riešenia. Dolná hranica ceny optimálneho riešenia je H (vychádzame z toho, že váhy všetkých položiek sú $v_i \leq V$). Relatívna chyba aproximatívneho algoritmu je teda nanajvýš $\epsilon = \frac{n2^b}{H}$. Ak teda chceme skonštruovať ϵ–aproximatívny algoritmus pre *ľubovoľné*, dané $\epsilon > 0$, stačí zvoliť $b = \lceil \log \frac{\epsilon H}{n} \rceil$. Zložitosť algoritmu bude $\mathcal{O}(\frac{n^2 H}{2^b}) = \mathcal{O}(\frac{n^3}{\epsilon})$ a teda polynomiálna vzhľadom k dĺžke prípadu a k $\frac{1}{\epsilon}$.

8.3 Redukcie

Úplná aproximatívna schéma je evidentne najlepšie možné riešenie optimalizačného problému v prípade, keď polynomiálny algoritmus poskytujúci jeho presné riešenie nie je známy. Preto je pre NP–ťažké optimalizačné problémy dôležité vedieť, či sa pre ne dá skonštruovať úplná aproximatívna schéma resp. aproximatívna schéma resp. relatívny aproximatívny algoritmus. Keďže odpovede na otázky takéhoto typy sa hľadajú veľmi ťažko, ponúka sa riešenie obdobné tomu, ktoré sme použili pri úvahách o triede NP—redukcie. Redukcia medzi problémami nám umožní jednak určiť relatívnu obtiažnosť problémov, jednak je nástrojom dovoľujúcim získať výsledky o neaproximovateľnosti.

Pojem L–redukcie je analogický pojmu polynomiálnej reduckie. Jej kľúčovou vlastnosťou je, že ak R je polynomiálna redukcia problému A na problém B, tak riešenie prípadu $R(x)$ problému B je zároveň riešením prípadu x problému A. Ak chceme, aby aj redukcia medzi optimalizačnými problémami mala analogickú vlastnosť, tak musíme vziať do úvahy dve funkcie: funkciu R, ktorá prípadu x problému A priradí prípad $R(x)$ problému B, a funkciu S, ktorá riešeniu prípadu $R(x)$ priradí riešenie prípadu x, pričom obidve riešenia budú „rovnako kvalitné".

Definícia 8.18 *Nech A a B sú optimalizačné problémy toho istého typu.* L–redukcia *problému A na problém B je dvojica funkcií (R, S), ktoré sú vypočítateľné v polynomiálnom čase a ktoré majú naviac tieto dve vlastnosti:*

1. *Ak x je prípadom problému A s optimálnou cenou $OPT(x)$, tak $R(x)$ je prípadom problému B s optimálnou cenou vyhovujúcou nerovnosti*

$$OPT(R(x)) \leq \alpha \cdot OPT(x) ,$$

   kde α je kladná konštanta.

2. *Ak s je ľubovoľné prípustné riešenie prípadu $R(x)$ problému B, tak $S(s)$ je prípustné riešenie prípadu x problému A také, že*

$$|OPT(x) - c(S(s))| \leq \beta \cdot |OPT(R(x)) - c(s)|,$$

   kde β je ďalšia kladná konštanta a c označuje cenu riešenia.

Funkcia S zaručuje, že získané riešenie prípadu x nebude horšie (viac vzdialené od optimálneho) ako riešenie prípadu $R(x)$. Naviac sa zachováva optimalita: ak s je optimálne riešenie prípadu $R(x)$, tak $S(s)$ je optimálne riešenie prípadu x.

Príklad 8.19 *Triviálna redukcia problému NEZÁVISLÁ MNOŽINA na problém VRCHOLOVÉ POKRYTIE (R je identická funckia a S vráti pre množinu M jej komplement $V - M$) nie je L–redukciou. Príčina je v tom, že optimálne riešenie problému vrcholového pokrytia môže byť neohraničene väčšie ako optimálne riešenie problému nezávislej množiny (stačí uvážiť graf obsahujúci veľkú kliku).*

Keď sa však obmedzíme na grafy stupňa nanajvýš k, tak popísaná dvojica funkcií (R,S) je L–redukciou. Ak maximálny stupeň vrchola grafu je k, tak maximálna nezávislá množina má aspoň $\frac{|V|}{k+1}$ vrcholov, zatiaľ čo minimálne vrcholové pokrytie má nanajvýš $|V|$ vrcholov. Preto stačí zvoliť konštantu $\alpha = k + 1$. Rozdiel medzi ľubovoľným pokrytím C a optimálnym pokrytím je taký istý ako medzi množinou $V - C$ a maximálnou nezávislou množinou a preto stačí vziať $\beta = 1$.

L–redukcia má podobné vlastnosti ako polynomiálna redukcia definovaná v 5.1.

Lema 8.20 *Ak (R, S) je L–redukcia problému A na problém B a (R', S') je L–redukcia problému B na problém C, tak ich kompozícia $(R \cdot R', S \cdot S')$ je L–redukciou problému A na problém C.*

Dôkaz Kompozície funkcií $R \cdot R'$ a $S \cdot S'$ sú vypočítateľné v polynomiálnom čase. Ďalej pre prípad x problému A máme $OPT(x) \leq \alpha \cdot OPT(R(x))$ a $OPT(R(x)) \leq \alpha' \cdot OPT(R'(R(x)))$, kde α a α' sú zodpovedajúce konštanty. Preto $OPT(x) \leq \alpha \cdot \alpha' \cdot OPT(R'(R(x)))$ a $R \cdot R'$ vyhovuje prvej podmienke s konštantou $\alpha \cdot \alpha'$. Podobne pre druhú kompozíciu. □

Kľúčovou vlastnosťou L–redukcie je, že zachováva aproximatívnosť:

Lema 8.21 *Ak existuje L–redukcia (R, S) problému A na problém B s konštantami α a β, a naviac existuje polynomiálny ϵ–aproximatívny algoritmus pre B, tak potom existuje aj polynomiálny $\frac{\alpha\beta\epsilon}{1-\epsilon}$ –aproximatívny algoritmus pre A.*

Dôkaz Aproximatívny algoritmus pre problém A je takýto. Pre daný prípad x problému A, vypočítame prípad $R(x)$ problému B. Naň aplikujeme predpokladaný ϵ–aproximatívny algoritmus pre B; získame riešenie s. Nakoniec vypočítame riešenie $S(s)$ prípadu x problému A.

Vypočítame relatívnu chybu navrhnutého algoritmu, t.j. podiel

$$\frac{|OPT(x) - c(S(s))|}{\max\{OPT(x), c(S(s))\}}.$$

Podľa druhej vlastnosti L–redukcie vieme, že

$$|OPT(x) - c(S(s))| \leq \beta \cdot |OPT(R(x)) - c(s)|.$$

Z prvej vlastnosti L–redukcie dostávame, že

$$\max\{OPT(x), c(S(s))\} \geq \frac{OPT(R(x))}{\alpha} \geq \frac{\max\{OPT(R(x)), c(s)\}}{\alpha}(1-\epsilon).$$

Dosadením získame

$$\frac{|OPT(x) - c(S(s))|}{\max\{OPT(x), c(S(s))\}} \leq \frac{\alpha\beta}{1-\epsilon} \cdot \frac{|OPT(R(x)) - c(s)|}{\max\{OPT(R(x)), c(s)\}} \leq \frac{\alpha\beta}{1-\epsilon} \cdot \epsilon,$$

lebo na riešenie problému B sme použili ϵ–aproximatívny algoritmus. Dostávame hľadaný odhad relatívnej chyby algoritmu pre A. □

Dôležitou vlastnosťou navrhnutého aproximatívneho algoritmu pre problém A je, že keď hodnota čísla ϵ je kladná a blíži sa k nule, tak aj hodnota výrazu $\frac{\alpha\beta\epsilon}{1-\epsilon}$ sa blíži k nule, tj. s rastúcou kvalitou aproximatívneho algoritmu pre B rastie aj kvalita aproximatívneho algoritmu pre A. Ak existuje L–redukcia problému A na problém B a naviac pre problém B existuje polynomiálna aproximatívna schéma, tak potom existuje polynomiálna aproximatívna schéma aj pre problém A.

8.4 Neaproximovateľnosť

Zhrňme výsledky predchádzajúcich sekcií: definovali sme päť tried optimalizačných problémov:

NPO optimalizačné problémy, ktorých asociované jazyky patria do NP.

APX problémy z NPO, pre ktoré existuje ϵ–aproximatívny algoritmus.

PAS problémy z NPO, pre ktoré existuje aproximatívna schéma.

FPAS problémy z NPO, pre ktoré existuje úplná aproximatívna schéma.

PO problémy z NPO, ktoré sú riešiteľné deterministicky v polynomiálnom čase.

Zrejme platí

$$\mathsf{PO} \subseteq \mathsf{FPAS} \subseteq \mathsf{PAS} \subseteq \mathsf{APX} \subseteq \mathsf{NPO}.$$

Úplná aproximatívna schéma je evidentne najlepšie možné riešnie optimalizačného probému v prípade, keď polynomiálny algoritmus poskytujúci jeho presné riešenie nie je známy. Ukážeme, že existujú problémy, pre ktoré, za predpokladu $\mathsf{P} \neq \mathsf{NP}$, *neexistuje* žiadna úplná chéma, tj. že za predpokladu $\mathsf{P} \neq \mathsf{NP}$ je $\mathsf{APX} \neq \mathsf{NPO}$. Analogickú otázku si môžeme položiť aj pre polynomiálne aproximatívne schémy a pre relatívne aproximatívne algoritmy. Ukážeme, že za predpokladu $\mathsf{P} \neq \mathsf{NP}$ existujú problémy, pre ktoré sa nedá zostrojiť aproximatívna schéma a ani aproximatívny algoritmus.

Negatívne výsledky pre relatívnu aproximáciu

Problémom, pre ktorý je dokážeme negatívny výsledok je problém obchodného cestujúceho.

PROBLÉM OBCHODNÝ CESTUJÚCI (OC)
VSTUP: Orientovaný graf $G = (V, H)$, v ktorom každej hrane je priradená dĺžka $w : V \to \mathbb{N}$. Prípustným riešením je cyklus v grafe, ktorý navštívi každý vrchol práve raz. Cenou riešenia je dĺžka cyklu
ÚLOHA: Nájsť riešenie s minimálnou cenou.

Veta 8.22 *Ak* $\mathsf{P} \neq \mathsf{NP}$*, tak pre problém* OBCHODNÝ CESTUJÚCI *neexistuje žiaden* ϵ*–aproximatívny algoritmus pre* $\epsilon < 1$.

Dôkaz Predpokladajme, existenciu ϵ–aproximatívneho algoritmu E pre problém OC. Využijúc tento algoritmus, navrhneme polynomiálny algoritmus A na riešenie NP–úplného problému HAMILTONOVSKÝ CYKLUS[2]. Algoritmus A dostane na vstup graf $G = (V, H)$. Vytvorí nový, úplný ohodnotený graf G' s množinou vrcholov V. Hrana (i, j) grafu G' má dĺžku 1 práve ak $(i, j) \in H$ a má dĺžku $\frac{|V|}{1-\epsilon}$ práve ak $(i, j) \notin H$. Na graf G' sa môžeme pozerať ako na prípad problému OC. Pritom pre jeho optimálne riešenie platí:

$$OPT(G') = |V| \iff G \text{ má hamiltonovský cyklus}$$

$$OPT(G') \geq \frac{|V|}{1-\epsilon} + |V| - 1 \iff G \text{ nemá hamiltonovský cyklus} .$$

Algoritmus A ďalej simuluje ϵ–aproximatívny algoritmus E (pre problém OC) na vstupe G'. Ak $OPT(G') = |V|$, tak $c(E(G')) \leq \frac{|V|}{1-\epsilon}$. Naopak, ak $OPT(G') \geq \frac{|V|}{1-\epsilon} + |V| - 1$, tak $c(E(G')) > \frac{|V|}{1-\epsilon}$. Preto cena riešenia poskytnutého algoritmom E jednoznačne určuje, či G má alebo nemá hamiltonovský cyklus. Algoritmus A poskytne ako výstup zodpovedajúcu odpoveď.

□

Upozorňujeme, že v predchádzajúcom dôkaze sme použili špeciálny druh redukcie medzi problémami. V uvažovanom príklade existuje veľký rozdiel medzi optimálnou cenou ak originálny prípad je "áno" prípadom problému hamiltonovského cyklu a optimálnou cenou ak originálny prípad je "nie" prípadom. Naviac sa ukázalo, že aproximatívny algoritmus je citlivý na tento rozdiel.

Poznamenávame, že existujú ϵ–aproximatívne algoritmy pre rôzne špeciálne prípady problému obchodného cestujúceho. Napríklad ak vzdialenosti medzi mestami môžu byť len 1 alebo 2, tak existuje $\frac{1}{7}$–aproximatívny algoritmus. Pre prípad, že vzdialenosti medzi mestami spĺňajú trojuholníkovú nerovnosť, je známy $\frac{1}{3}$–aproximatívny algoritmus.

[2]Úlohou je rozhodnúť, či v grafe existuje cyklus obsahujúci všetky vrcholy.

Neexistencia aproximatívnej schémy

Podrobnejšie preskúmame otázku, či pre daný optimalizačný problém existuje polynomiálna aproximatívna schéma. Na to, aby sme mohli ukázať výsledok tohoto typu, vyjdeme z alternatívnej charakterizácie triedy NP, ktorú sme prezentovali v časti 6.1.

Podľa lemy 6.1 jazyk L patrí do NP práve vtedy, keď preň existuje overovateľ polynomiálnej časovej zložitosti. Ako ukážeme ďalej, požiadavku na polynomiálnu časovú zložitosť overovateľa možno ešte viac oslabiť.

Lema 8.23 *Jazyk $L \in$ NP vtedy a len vtedy, ak preň existuje deterministický overovateľ s logaritmickou priestorovou zložitosťou.*

Dôkaz Vyjdeme z NP–úplnosti problému splniteľnosti CNF. Nech F je redukcia jazyka L na CNF. Definujeme reláciu R takto: $(x, y) \in R$ práve vtedy, ak $y = \langle F(x), z\rangle$, kde z je spĺňajúce priradenie hodnôt premenných formuly $F(x)$. Zrejme $x \in L$ práve ak existuje y také, že $(x, y) \in R$. Naviac, relácia R sa dá rozhodovať v logaritmickom priestore: stroj, ktorý počíta redukciu $F(x)$ porovná (bit po bite) svoj výstup s reťazcom y. Overenie, či z spĺňa $F(x)$ sa dá v logaritmickom priestore realizovať triviálne. □

Dôležitým poznatkom je, že zložitosť overovateľa je možné redukovať ešte viac. Overovateľ, s ktorým budeme v ďalšom pracovať, je pravdepodobnostný stroj, ktorý pri overovaní, či $(x, y) \in R$ môže použiť $\mathcal{O}(\log |x|)$ náhodných bitov. Naviac, aj keď má ničím neohraničený prístup k reťazcu x, jeho *prístup k reťazcu y je veľmi ohraničený*: má prístup len ku konštantnému počtu bitov reťazca y. Realizácia je nasledovná. Pre vstup x, využijúc náhodný reťazex $r \in \{0, 1\}^{\lceil c \log |x| \rceil}$, pre nejaké $c > 0$, overovateľ vypočíta množinu čísiel $Q(x, r) = \{i_1, \ldots , i_k\}$, kde k je fixované číslo a všetky hodnoty i_j sú nanajvýš $|y| = p(|x|)$. Potom i_j–ty symbol reťazca y sa objaví na overovateľovej páske, pre $j = 1, \ldots , k$; zvyšok reťazca y je overovateľovi neprístupný. Nakoniec overovateľ, poznajúc $x, r, y_{i_1}, \ldots , y_{i_k}$, vykoná polynomiálne časovo ohraničený výpočet.

Overovateľ uvedených vlastností nazývame $(\log n, 1)$*–ohraničený overovateľ*, aby sme zachytili jeho dve dôležité ohraničenia: $\mathcal{O}(\log n)$ náhodných bitov a konštantný (t.j. $\mathcal{O}(1)$) počet bitov reťazca y. Hovoríme, že $(\log n, 1)$–ohraničený overovateľ rozhoduje reláciu R, ak pre každú vstupnú dvojicu reťazcov (x, y) platí: ak $(x, y) \in R$, tak pre všetky náhodné reťazce r skončí výpočet overovateľa v akceptujúcej konfigurácii. Ale ak $(x, y) \notin R$, tak aspoň pre $\frac{1}{2}$ spomedzi všetkých náhodných reťazcov výpočet skončí v zamietajúcej konfigurácii. Inak povedané, pripúšťame, aby ohraničená časť pozitívnych výsledkov bola nesprávna; všetky negatívne výsledky sú správne. Nová charakterizácia triedy NP je vyjadrená v nasledovnej vete.

Veta 8.24 *Jazyk $L \in$ NP vtedy a len vtedy, ak preň existuje $(\log n, 1)$–ohraničený overovateľ.*

Pre uvedenú vetu sa používa názov *PCP teoréma*, ktorý vznikol skrátením z **P**robabilistically **C**heckable **P**roofs. Má bezprostredné dôsledky týkajúce sa aproximovateľnosti optimalizačných problémov.

Ak $(\log n, 1)$–ohraničený overovateľ dostane vstup (x, y), $|x| = n$, tak existuje práve $2^{\mathcal{O}(\log n)} = n^{\mathcal{O}(1)}$ možností výberu pre náhodný reťazec. Ďalej, akonáhle je fixovaný náhodný reťazec, tak priebeh výpočtu závisí len na $\mathcal{O}(1)$ bitoch reťazca y. Presnejšiu formuláciu poskytuje nasledujúca Lema.

Lema 8.25 *Nech pre jazyk L existuje $(\log n, 1)$–ohraničený overovateľ. Potom existujú celé čísla $c, d, k \geq 1$ také, že pre každý vstup x dĺžky n existuje n^c booleovských funkcií $f_1, f_2, \ldots$ n^d premenných s nasledujúcimi vlastnosťami:*

1. *Každá funkcia f_i je funkciou k premenných a pre dané (x, i) sa tabuľka pravdivostných hodnôt funkcie f_i dá vypočítať v polynomiálnom čase.*

2. *Ak slovo x patrí do L, tak existuje také priradenie hodnôt premenným, že každá funkcia f_i je splnená.*

3. *Ak slovo x nepatrí do L, tak pre žiadne priradenie hodnôt premenným nie je splnených viac než $1/2$ funkcií f_i.*

Dôkaz Majme jazyk L a $(\log n, 1)$–ohraničený overovateľ V rozhodujúci reláciu R takú, že $L = \{\, x \mid (x, y) \in R \text{pre nejaké } y \,\}$. Bez straty na všeobecnosti môžeme predpokladať, že $x, y \in \{0,1\}^*$. Predpokladajme, že V pri overovaní, či $(x, y) \in R$ používa $\lceil c \log |x| \rceil$ náhodných bitov. To implikuje, že pre daný vstup (x, y), kde $|x| = n$, existuje nanajvýš $2^{\lceil c \log n \rceil} \leq 2n^c$ rôznych výpočtov; v každom sa prečíta nanajvýš $\mathcal{O}(1)$ bitov reťazca y. Bez straty všeobecnosti preto môžememe predpokladať, že dĺžka reťazca y je $\mathcal{O}(n^c)$, presnejšie n^d pre nejaké celé číslo d.

Ak fixujeme náhodný reťazec $r \in \{0,1\}^{\lceil c \log n \rceil}$, máme tým presne určené, ktoré bity reťazca $y = y_1 y_2 \dots y_{n^d}$ overovateľ pozná. Nech počet týchto bitov je nanajvýš k (k konštanta). Nech ich pozície sú $i_1(r), \dots i_k(r)$. Ďaľší priebeh výpočtu overovateľa závisí od toho, aké sú hodnoty $y_{i_1(r)}, \dots y_{i_k(r)}$. Definujme booleovskú funkciu k premenných, f_r, takú, že $f_r(b_1, \dots, b_k) = 1$ vtedy a len vtedy, ak overovateľ akceptuje vstup v prípade, že premenné $y_{i_1(r)}, \dots y_{i_k(r)}$ majú hodnoty $b_1, \dots, b_k$. Keďže výpočet overovateľa je polynomiálne časovo ohraničený, tak tabuľku pravdivostných hodnôt funkcie f_r (obsahuje hodnoty funkcie f_r pre všetkých 2^k rôznych vektorov $b_1, \dots, b_k$) vieme vypočítať v polynomiálnom čase.

Uvažujeme teraz množinu všetkých funkcií $\left\{\, f_r \;\middle|\; r \in \{0,1\}^{\lceil c \log n \rceil} \,\right\}$ definovaných uvedeným spôsobom. Podľa definície $(\log n, 1)$–ohraničeného overovateľa ak slovo x patrí do jazyka L, tak existuje taký reťazec $y = y_1 y_2 \dots y_{n^d}$, že všetky funkcie z množiny sú splnené. V opačnom prípade žiaden reťazec y nemôže spôsobiť, že viac ako $1/2$ funkcií z množiny bude splnených. □

Veta 8.26 *Ak existuje polynomiálna aproximatívna schéma pre MAXCNF*[3], *tak* $\mathsf{P} = \mathsf{NP}$.

Dôkaz Nech L je NP–úplný jazyk a nech x je slovo, o ktorom chceme rozhodnúť, či patrí do L. Predpokladajme, že máme polynomiálnu aproximatívnu schému pre optimalizačný problém MAXCNF, ktorá poskytuje aproximáciu s relatívnou chybou ϵ v čase $p_\epsilon(n)$, p_ϵ polynóm. Na základe tohoto predpokladu skonštruujeme deterministický polynomiálny algoritmus rozhodujúci jazyk L.

Podľa vety 8.24 existuje pre L $(\log n, 1)$–ohraničený overovateľ. Využijeme notáciu zavedenú v leme 8.25. Nech $y_1, \dots, y_{n^d}$ je súbor booleovských premenných a $\{f_i \mid 1 \leq i \leq 2n^c\}$ množina booleovských funkcií, ktoré zodpovedajú slovu x.

Uvažujme funkiu f_i z tejto množiny. Nech je to funkcia premenných $y_{i_1}, \dots, y_{i_k}$. Potom f_i môžeme vyjadriť ako konjunkciu maximálne 2^k klauzúl veľkosti k a obsahujúcich uvedené premenné. Označme klauzuly $C_{i,1}, \dots, C_{i,2^k}$. Potom formula

$$F = \bigwedge_{i=1}^{n^c} \bigwedge_{j=1}^{2^k} C_{i,j}$$

v konjunktívnom normálnom tvare je splniteľná, práve ak $x \in L$. Naviac, ak $x \notin L$, tak pre každý reťazec y aspoň polovica funkcií $f_i$nie je splnených; každá nesplnená funkcia obsahuje aspoň jednu nesplnenú klauzulu. Spolu, maximálny možný počet splnených klauzúl formuly F je buď $n^c 2^k$ alebo nanajvýš $2n^c 2^k - \frac{1}{2} 2n^c = (1 - \frac{1}{2^{k+1}}) 2n^c 2^k$.

Teraz využijeme predpokladanú polynomiálnu aproximatívnu schému pre MAXCNF. Aplikujeme schému s $\epsilon = \frac{1}{2^{k+2}}$ na formulu F. Časová zložitosť výpočtu bude $p_{1/2^{k+1}}(2n^c 2^k)$ a teda polynomiálna (2^k je konštatna!). Spojením faktov o možnom počte splnených klauzúl formuly F a veľkosti relatívnej chyby schémy dostávame: Ak výsledkom výpočtu schémy je, že existuje priradenie hodnôt premenným formuly F spĺňajúce aspoň $(1 - \frac{1}{2^{k+2}}) 2n^c 2^k$ klauzulí, tak $x \in L$; v opačnom prípade $x \notin L$. □

[3]Jedná sa o modifikáciu problému splniteľnosti. Pre danú booleovskú funkciu v konjunktívnom normálnom tvare je úlohou nájsť také priradenie hodnôt premenným, aby bol splnený maximálny možný počet klauzúl formuly.

Časť IV

Trieda P

Kapitola 9

Paralelné výpočty

Dôsledkom technického pokroku posledných desaťročí je možnosť konštruovať masívne paralelné počítače, t.j. počítače pozostávajúce z obrovského množstva procesorov, schopných pracovať paralelne na riešení jedného problému. V tom istom čase sa začína formovať aj teória paralelných výpočtov, cieľom ktorej je charakterizácia problémov vhodných pre riešenie na paralelných počítačoch. Táto časť teórie zložitosti veľmi úzko súvisí s praktickými aplikáciami. Predmetom skúmania nie sú všetky algoritmicky riešiteľné problémy, ale len tie problémy z P, ktoré sa dajú efektívne riešiť paralelne.

9.1 Paralelné algoritmy

Hlavným cieľom pri návrhu paralelných algoritmov je minimalizovať čas medzi začiatkom a koncom všetkých paralelných výpočtov. Naviac by sme chceli, aby paralelné algoritmy boli *podstatne* (podľa možnosti exponenciálne) rýchlejšie ako sekvenčné, pretože to je hlavnou motiváciou ich konštrukcie. Samozrejme, že paralelné algoritmy nemôžu vyžadovať enormné množstvo procesorov.

V snahe porozumieť možnostiam, ktoré poskytuje paralelizmus, ale aj problémom, ktoré so sebou prináša, uvedieme najprv neformálne niekoľko paralelných algoritmov. Sú navrhnuté pre paralelný počítač s veľkým množstvom navzájom nezávislých procesorov. Každý procesor vykonáva svoj vlastný program. Komunikácia medzi procesormi je okamžitá a synchronizovaná; realizuje sa cez spoločne zdieľanú pamäť. To znamená, že všetky procesory súčasne vykonajú svoju prvú inštrukciu, vymenia si informácie, vykonajú druhú inštrukciu atď.

Príklad 9.1 (TRIEDENIE) *Je daných n položiek; úlohou je utriediť ich podľa kľúčov.*

Prvá verzia hľadaného paralelného algoritmu bude veľmi jednoduchá. Rozdelíme n položiek rovnomerne medzi procesory, každú podmnožinu utriedime jedným procesorom a utriedené podpostupnosti spojíme.

Vyjadrime počet procesorov použitých vo výpočte ako $p(n)$, t.j. ako funkciu počtu položiek n. Predpokladajme, že položky, ktoré máme utriediť sú na začiatku výpočtu v spoločnej pamäti na pamäťových miestach $x_1, \ldots, x_n$. V prvej fáze algoritmu sa každému procesoru priradí rovnako veľa položiek. V druhej fáze každý procesor (paralelne) utriedi svoju podmnožinu nejakým dobrým sekvenčným triediacim algoritmom; vyžiada si to čas $\mathcal{O}((n/p(n))\log(n/p(n)))$. V tretej fáze výpočtu potrebujeme spojiť všetky utriedené podpostupnosti do jednej utriedenej postupnosti. Táto časť výpočtu vychádza z nasledovnej myšlienky: postupne budeme (súčasne) spájať vždy dve podpostupnosti. V prvej etape použijeme $p(n)/2$ procesorov a spojíme $p(n)/2$ dvojíc podpostupností veľkosti $n/p(n)$. V ďalšej etape použijeme $p(n)/4$ procesorov a spojíme $p(n)/4$ dvojíc podpostupností veľkosti $2n/p(n)$. V záverečnej etape použitím jedného procesora spojíme dve podpostupnosti veľkosti $n/2$. Každá etapa sa realizuje sekvenčne. Časovo najnáročnejšia je zrejme posledná, ktorá si vyžaduje čas $\mathcal{O}(n)$.

Sčítaním časovej zložitosti všetkých fáz výpočtu dostávame asymptotický odhad časovej *zložitosti paralelného triedenia využívajúceho $p(n) \leq n$ procesorov:*

$$\mathcal{O}(n + \frac{n}{p(n)} \log \frac{n}{p(n)}).$$

Ak počet procesorov je konštantný (nezávislý od n), tak dostávame paralelný algoritmus zložitosti $\mathcal{O}(n \log n)$. Ak počet procesorov je malý, presnejšie ak $p(n) \leq \log n$, tak paralelná zložitosť je $\mathcal{O}(n(\log n)/p(n))$. Na druhej strane ak počet procesorov presiahne $\log n$, tak paralelná zložitosť je $\mathcal{O}(n)$.

Celkové zrýchlenie dosiahnuté využitím paralelizmu však nie je príliš veľké: od času $n \log n$ sme sa dostali k času n. Optimálne by bolo, keby sme dosiahli exponenciálne urýchlenie, to znamená logaritmický paralelný čas $\mathcal{O}(\log n)$ alebo aspoň polylogaritmický čas $\mathcal{O}((\log n)^k)$ (k je konštanta). V prípade triedenia je takéto zlepšenie možné.

Podstatou triedenia je určiť pozíciu (rank) každej položky v utriedenej postupnosti. Ak položky $x_1, \ldots, x_n$ majú rank $r_1, \ldots, r_n$, tak v jednom paralelnom kroku n procesorov môže preusporiadať celú postupnosť tak, že i–ty procesor vykoná priradenie $x_i := x_{r_i}$ (predpokladáme, že všetky položky sú navzájom rôzne).

Problémom je vypočítať efektívne rank každej položky. Ak rank jednej položky dokážeme vypočítať nezávisle od ostatných, tak paralelný čas potrebný na výpočet ranku celej postupnosti bude zhodný s časom na výpočet ranku jednej položky.

Ukážeme, ako sa dá efektívne vypočítať rank celej postupnosti využitím n^2 procesorov. V prvej fáze výpočtu sa každému procesoru priradí dvojica (i, j), kde $1 \leq i, j \leq n$. Procesor (i, j) určí, či položka x_i je väčšia alebo rovná ako x_j; ak áno, tak na pozíciu c_{ij} uloží hodnotu 1, inak hodnotu 0. V druhej fáze výpočtu procesory $(i, 1), \ldots, (i, n)$ vypočítajú sumu $\sum_{j=1}^{n} c_{ij}$. Suma $\sum_{j=1}^{n} c_{ij}$ je rankom položky x_i a uloží sa do r_i. Pritom suma sa dá vypočítať paralelne v čase $\mathcal{O}(\log n)$, využívajúc adičný strom.

Tým sme skonštruovali nový paralelný triediaci algoritmus využívajúci n^2 procesorov a s časovou zložitosťou $\mathcal{O}(\log n)$[1]. Dosiahli sme exponenciálne urýchlenie v porovnaní so sekvenčným algoritmom. Dôležité pritom je, že počet použitých procesorov je polynomiálny. Exponenciálny počet procesorov je totižto ešte menej realistickým predpokladom než exponenciálny čas.

Príklad 9.2 (NÁSOBENIE MATÍC) *Majme dané dve matice A a B rozmerov $n \times n$. Úlohou je vypočítať ich súčin, teda maticu $C = A \cdot B$, kde*

$$C_{ij} = \sum_{k=1}^{n} A_{ik} B_{kj}, \quad i, j = 1, \ldots, n.$$

Triviálny sekvenčný algoritmus na násobenie matíc zahŕňa $\mathcal{O}(n^3)$ aritmetických operácií sčítania a násobenia. Je to zároveň algoritmus, ktorý sa veľmi jednoducho a priamočiaro implementuje ako paralelný. Všetkých n^3 súčinov sa vypočíta súčasne v n^3 procesoroch; nazvime (i, j, k) procesor počítajúci súčin $A_{ik} B_{kj}$. V druhej fáze výpočtu usporiadame procesory tak, že vždy n procesorov vypočíta súčet n súčinov v čase $\log n$ (opäť využitím adičného stromu).

Celkový paralelný čas potrebný na násobenie matíc je tak $1 + \log n$ a počet procesorov n^3. Dosiahnuté zrýchlenie je, podobne ako v predchádzajúcom príklade, exponenciálne. Otázkou je, či navrhnutý algoritmus je naozaj najlepší možný. Nie je ťažké overiť, že $\log n$ paralelných krokov je nevyhnutných na násobenie dvoch matíc. Ako je to s počtom procesorov?

Namiesto toho, aby sme počítali všetkých n^3 súčinov v jedinom kroku, vypočítame ich postupne v $\log n$ krokoch, v každom kroku využívajúc $\lceil n^3 / \log n \rceil$ procesorov. V druhej fáze, podobne ako v pôvodnom algoritme, počítame n^2 súm. Ak by sme v niektorom kroku potrebovali viac ako $\lceil n^3 / \log n \rceil$ procesorov, nahradíme opäť tento krok postupnosťou krokov (konkrétne sa to týka prvých $\log \log n$ adičných krokov). Tak dostávame paralelný algoritmus s časovou zložitosťou neprevyšujúcou $3 \log n$ a počtom procesorov $\lceil n^3 / \log n \rceil$.

Uvedený príklad je dôležitý pre to, že ukazuje aplikáciu tzv. *Brentovho princípu* (Brent Scheduling Principle). Je to technika, umožňujúca optimalizovať počet potrebných paralelných procesorov:

[1] Prezentovaný algoritmus nie je najlepší známy!

Veta 9.3 *Ak existuje paralelný algoritmus s časovou zložitosťou $t(n)$ a používajúci $p(n)$ procesorov, tak pre funkciu $q(n) < p(n)$ existuje modifikovaný algoritmus na riešenie toho istého problému s časovou zložitosťou $\Theta(\frac{p(n)t(n)}{q(n)})$ a používajúci len $q(n)$ procesorov.*

Dôkaz Podobne ako v konštrukcii naznačenej pri riešení problému násobenia matíc, rozdelíme p procesorov na q skupín a použijeme q procesorov, ktoré máme k dispozícii, na simuláciu každej skupiny. V prvom kroku modifikovaného algoritmu simuluje každý z q procesorov prvý krok prvého procesoru svojej skupiny. Ďalej simuluje prvý krok druhého procesoru a tak ďalej. Pretože každá skupina obsahuje nanajvýš $\lceil p/q \rceil$ procesorov, vyžiada si simulácia prvého kroku všetkých procesorov modifikovaným algoritmom $\lceil p/q \rceil$ krokov. Celková časová zložitosť modifikovaného algoritmu je $\Theta(\lceil p/q \rceil t)$. Ak uvážime, že $p/q \leq \lceil p/q \rceil < 2p/q$, tak získame tvrdenie vety. □

Skutočnosť, že doteraz sme v paralelných algoritmoch uvažovali počet procesorov ako funkciu dĺžky vstupu, sa zdá dosť neprirodzená. Reálne počítače majú fixovaný počet procesorov a na nich musíme riešiť prípady akéhokoľvek rozsahu. Čo sa napríklad stane, ak máme počítať súčin dvoch $n \times n$ matíc na počítači s P procesormi, kde P je podstatne menšie než $\lceil n^3/\log n \rceil$? Vychádzajúc z navrhnutého algoritmu a aplikáciou Brentovho princípu môžeme ho modifikovať tak, aby sa dal realizovať za použitia P procesorov. Uvedená úvaha zdôrazňuje význam snahy o minimalizáciu počtu procesorov používaných paralelnými algoritmami: ak v základnom algoritme sa používa príliš veľký počet procesorov, odrazí sa to následne na vysokej časovej zložitosti pri použití tohoto algoritmu na riešenie prípadov veľkého rozsahu na počítači s fixným počtom procesorov.

Príklad 9.4 (MAXIMÁLNY TOK) *Problém nájdenia maximálneho toku v grafe poukazuje na limity paralelného prístupu k riešeniu problémov. Sekvenčný algoritmus hľadá optimálny tok v grafe postupnými iteráciami. Prvá iterácia je nulový tok. Každú ďaľšiu iteráciu získame tak, že sa snažíme zmeniť ohodnotenie niektorej hrany v súlade s jej kapacitou a tak, aby sa zvýšil celkový tok. Ak takúto zmenu nie je možné previesť, našli sme maximálny tok.*

Ľahko sa skonštruuje paralelný algoritmus na výpočet jednej iterácie. Problémom je, že iterácie musia byť počítané postupne, jedna za druhou. Samozrejme, je možné pokúsiť sa spojiť výpočet niekoľkých iterácii do jednej paralelnej fázy, resp. postaviť paralelný algoritmus pre problém MAXIMÁLNY TOK na úplne iných princípoch. Ale ani takýto prístup zatiaľ neuspel a MAXIMÁLNY TOK zostáva príkladom problému, pre ktorý nie je známy žiaden polylogaritmický paralelný algoritmus.

Predchádzajúce príklady naznačili dve otázky týkajúce sa zložitosti paralelných výpočtov: kedy môžeme paralelný výpočet považovať za optimálny a kedy môžeme hovoriť o optimálnom urýchlení v porovnaní so sekvenčným algoritmom.

Aby sme mohli zodpovedať na obe formulované otázky, definujeme *cenu* (paralelného alebo sekvenčného) algoritmu ako súčin počtu použitých procesorov a času potrebného na vykonanie algoritmu. Ľahko vidíme, že táto hodnota je zároveň časom potrebným na simuláciu paralelného výpočtu na jedinom procesore (tj. sekvenčne), ktorý postupne simuluje jeden krok každého procesora. Nech A a B sú dva algoritmy pre ten istý problém s cenou c_A resp. c_B. Potom hovoríme, že A je *efektívny vzhľadom k B* ak $c_A = \mathcal{O}(c_B)$. Paralený algoritmus je *optimálny*, ak je efektívny vzhľadom k najlepšiemu možnému sekvenčnému algoritmu. V prípade, že je efektívny vzhľadom k najlepšiemu *známemu* sekvenčnému algoritmu, tak hovoríme o *optimálnom urýchlení*.

Často je možné redukovať časovú zložitosť paralelného algoritmu tým, že použijeme väčší počet procesorov a naopak. Obecne sa ovšem preferujú algoritmy s nižšou časovou zložitosťou: cieľom je návrh takých paralelných algoritmov, ktoré sú podstatne (tj. exponenciálne) rýchlejšie v porovnaní so sekvenčnými a pritom používajú rozumný (tj. polynomiálny) počet procesorov. Ako *efektívny* sa preto označuje paralelný algoritmus, ktorý si na riešenie prípadu rozsahu n vyžaduje $n^{\mathcal{O}(1)}$ procesorov a polylogaritmický čas $(\log n)^{\mathcal{O}(1)}$.

Zhrnutím dostávame nasledujúcu klasifikáciu problémov:

- *paralelne riešiteľné problémy*, pre ktoré existuje paralelný algoritmus s polynomiálnou časovou zložitosťou a využívajúci polynomiálny počet procesorov,

- *vysoko paralelizovateľné problémy*, pre ktoré existuje efektvíny paralelný algoritmus (s polylogaritmickou časovou zložitosťou a využívajúci polynomiálny počet procesorov),
- *vnútorne (dedičné) sekvenčné problémy*, ktoré sú paralelne riešiteľné, ale nedajú sa riešiť žiadnym efektívnym paralelným algoritmom.

Zrejme trieda paralelne riešiteľných problémov je zhodná s triedou P problémov riešiteľných sekvenčne v polynomiálnom čase. Otvorenou otázkou je, či pre každý problém z triedy P je možné navrhnúť efektívny paralelný algoritmus. Inak formulované, pýtame sa, či existujú vnútorne sekvenčné problémy. V časti 9.3.1 ukážeme, ktoré problémy sú najpravdepodobnejšími kandidátmi na vnútorne sekvenčné problémy.

Druhým dôležitým dôsledkom predchádzajúcich úvah je, že (za predpokladu $\mathsf{P} \neq \mathsf{NP}$) paralelizmus *nie je* spôsob, ako riešiť NP–úplné problémy. Každý paralelný algoritmus pre NP–úplný problém si vyžaduje buď exponenciálny čas alebo exponenciálny počet procesorov (alebo oboje). To samozrejme neznamená, že paralelné algoritmy sú bezcenné. V spojení s rýchlymi procesormi a efektívnymi programovacími technikami umožňujú exaktne riešiť stále väčšie a väčšie prípady NP–úplných problémov.

V nasledujúcej časti dáme predchádzajúcim (neformálnym) úvahám exaktný rámec tým, že presne definujeme paralelný výpočtový model, jeho zložitostné miery a následne aj triedu vysoko paralelizovateľných problémov.

9.2 Paralelné výpočtové modely

Faktom je, že výskum v teórii paralelných výpočtov sa nesústreďuje na jeden vybraný paralelný výpočtový model, ale na niekoľko tried výpočtových modelov. Za všetky spomeňme aspoň vektorové stroje, Booleovské (logické) obvody, paralelné RAMy, alternujúce Turingove stroje, agregáty, konglomeráty, neurónové siete a veľké množstvo strojov s fixovanou štruktúrou prepojenia (hyperkocky, stromy, shuffle–exchange). Každý z nich pritom odráža iné aspekty paralelného počítania: rozmiestnenie procesorov, ich vzájomné prepojenie, spôsob komunikácie medzi procesormi. Modely sa ďalej líšia typom inštrukcií (bitové inštrukcie vs. inštrukcie nad celými čísalmi). Jedným zo spôsobov, ako porovnať kvalitatívne charakteristiky rozmanitých modelov je ich vzájomná simulácia. Ak ľubovoľný stroj z triedy A sa dá simulovať strojom z triedy B, tak vieme, že každý problém riešiteľný strojom z triedy A je riešiteľný aj strojom z triedy B. Nás naviac zaujíma vzťah medzi zložitosťou výpočtov na strojoch z A a z B. Ak tieto sú zhruba rovnaké, môžeme oba modely považovať za v podstate ekvivalentné.

Našťastie sa ukazuje, že väčšina paralelných výpočtových modelov je polynomiálne ekvivalentná. Pojmy paralelne riešiteľného, vysoko paralelizovateľného a vnútorne dedičného problému sú tak invariantné vzhľadom k takto zvolenému výpočtovému modelu.

Nie všetky paralelné modely majú túto peknú vlastnosť. Napríklad niektoré modely s ohraničeným prepojením procesorov môžu byť príliš slabé na to, aby sa na nich dali efektívne simulovať iné modely (napr. stromové prepojenie). Na druhej strane iné modely sú príliš silné a nedajú s efektívne simulovať na väčšine bežných paralelných výpočtových modelov. Týka sa to strojov, ktoré dokážu generovať exponenciálne dlhé reťazce, resp. aktivovať exponenciálne veľa procesorov v polylogaritmickom čase.

Pretože v centre nášho záujmu je paralelná klasifikácia problémov z P, preferujeme model paralelný RAM (PRAM) v situáciach, keď nás zaujímajú horné odhady zložitosti a model logické obvody pre skúmanie dolných odhadov.

9.2.1 Stroje PRAM

Paralelný RAM je paralelným rozšírením stroja RAM. Pozostáva z potenciálne neohraničeného množstva strojov RAM $P_0, P_1, \ldots$ a z neohraničeného množstva spoločných (zdieľaných) pamäťových registrov $C_0, C_1, \ldots$. Každý procesor (stroj RAM) má svoju vlastnú, lokálnu, pamäť a pozná svoj index . V porovnaní so sekvenčným strojom RAM má naviac inštrukcie umožňujúce zápis do a čítanie zo spoločnej pamäte. Vstup je uložený na začiatku výpočtu v spoločnej pamäti, aby k nemu mali prístup všetky procesory. Všetky inštrukcie sú vykonávané v jednotkovom čase a synchronizovane.

K úplnej špecifikácii modelu PRAM musíme vyriešiť dva technické problémy. Prvý sa týka spôsobu, akým sa vo výpočte aktivizuje konečný počet spomedzi potenciálne nekonečného počtu procesorov. Jeden zo spôsobov je predpokladať, že procesor P_0 má špeciálny aktivačný register. Každý procesor s indexom menším než hodnota uložená v aktivačnom registri môže vykonávať svoj program. Na začiatku je aktívny len procesor P_0, všetky ostatné procesory čakajú. P_0 vypočíta aký počet procesorov je potrebných na riešenie zadaného vstupu; toto číslo uloží do aktivačného registra. Výpočet potom pokračuje až do toho okamihu, keď skončí výpočet procesora P_0; vtedy sa zastaví aj výpočet všetkých ostatných aktívnych procesorov.

Druhý technický problém sa týka práce so spoločnou pamäťou. Konflikt vzniká, keď niekoľko procesorov chce v tom istom okamihu zapisovať resp. čítať z toho istého pamäťového miesta. Podľa spôsobu, akým sa takýto konflikt rieši, rozlišujeme niekoľko variácií modelu PRAM.

- CRCW–PRAM (Concurrent–read concurrent–write PRAM)
  dovoľuje súčasne čítať z aj zapisovať na to isté pamäťové miesto. Potrebná je metóda riešiaca prípadné konflikty. Varianta PRIORITY predpokladá, že pri zápise uspeje procesor s najmenším indexom. Varianta COMMON predpokladá, že k zápisu dôjde len vtedy, ak všetky procesory, ktoré sa snažia zapísať do toho istého pamäťového registra, sa snažia zapísať to isté. Nakoniec varianta ARBITRARY predpokladá, že uspeje ľubovoľný procesor (upúšťame od determinizmu).

- CREW–PRAM (Concurrent–read exclusive–write PRAM)
  dovoľuje aby viacej procesorov súčasne čítalo z jedného pamäťového registra, ale vylučuje súčasný zápis.

- CROW–PRAM (Concurrent–read owner–write PRAM)
  vychádza z modelu CREW–PRAM . Predpokladá naviac, že každý pamäťový register má svojho vlastníka, ktorý ako jediný môže meniť jeho obsah.

- EREW–PRAM (Exclusive–read exclusive–write PRAM)
  vyžaduje, aby žiadne dva procesory nikdy nepristupovali súčasne k jednému pamäťovému registru.

Štandartným modelom v našich úvahách bude model CREW–PRAM s aktivačným registrom v procesore P_0. To znamená, že máme garantované, že procesor P_0 je aktívny po celú dobu trvania výpočtu a že najväčšia hodnota uložená v aktivačnom registri je hornou hranicou na počet použitých (aktívnych) procesorov.

Aby sme mohli s modelom pracovať, musíme ešte upresniť, ako sa zariadeniu zadávajú vstupné údaje, kde sa prečíta výstup a aká je cena výpočtu. Prijmeme nasledujúce konvencie:

Definícia 9.5 *Nech $\mathcal{M}$ je PRAM stroj.* Vstupno—výstupná konvencia *pre stroj $\mathcal{M}$ je nasledovná: Vstup $x \in \{0,1\}^n$ je zadávaný stroju $\mathcal{M}$ umiestnením čísla n do spoločného pamäťového registra C_0 a bitov $x_1, \ldots, x_n$ reťazca x do spoločných pamäťových registrov $C_1, \ldots, C_n$. $\mathcal{M}$ zobrazí svoj výstup $y \in \{0,1\}^m$ podobne: číslo m je v registri C_0 a bity $y_1, \ldots, y_m$ reťazca y v registroch $C_1, \ldots, C_m$.*

$\mathcal{M}$ počíta v paralelnom čase $t(n)$ *a s* počtom procesorov $p(n)$ *ak pre každý vstup $x \in \{0,1\}^n$ výpočet stroja $\mathcal{M}$ skončí po nanajvýš $t(n)$ krokoch a počas výpočtu je aktívnych nanajvýš $p(n)$ procesorov.*

Presný výber výpočtového modelu a definovanie jeho zložitostných mier nám umožňuje hovoriť o funkciách vypočítateľných v paralelnom čase $t(n)$ s počtom procesorov $p(n)$.

Definícia 9.6 *Nech f je funkcia $f : \{0,1\}^* \to \{0,1\}^*$. Funkcia $f(n)$ je vypočítateľná v* paralelnom čase $t(n)$ *a s* počtom procesorov $p(n)$ *ak existuje PRAM stroj $\mathcal{M}$, ktorý počíta v paralelnom čase $t(n)$ s počtom procesorov $p(n)$ a pre každý vstup x vypočíta $f(x)$.*

Pojmy paralelne riešiteľný a vysoko paralelizovateľný problém sú invariantné vzhľadom ku všetkým uvedeným variantám PRAM modelu. Preto každý z nich je vhodný na definovanie zložitostnej triedy NC, prezentovanej v časti 9.3. Avšak podtriedy NC^k triedy NC nie sú definované použitím modelu PRAM. Na ich definíciu sa využíva paralelný výpočtový model logických obvodov, tak ako je definovaný v časti 9.2.2. Dôvod je sčasti historický, sčasti je v tom, že trieda funkcií vypočítateľných v paralelnom čase $\mathcal{O}((\log n)^k)$ je rôzna pre rôzne varianty modelu PRAM.

9.2.2 Logické obvody

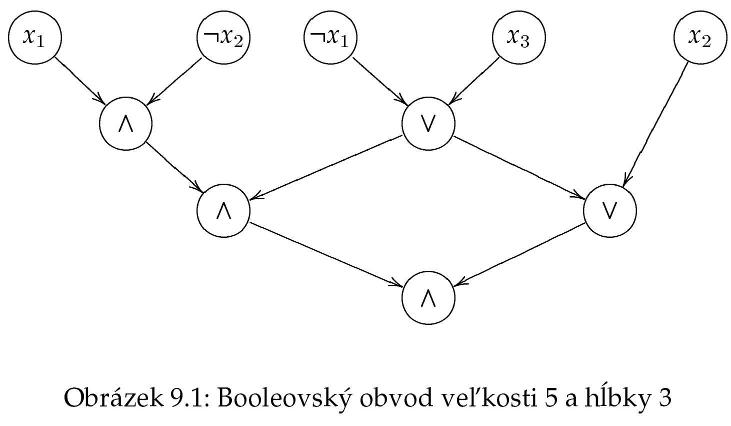
Model PRAM je prirodzeným paralelným rozšírením modelu RAM. Nie je ale úplne zrejmé, že takéto rozšírenie je naozaj zmysluplné. Zodpovedá PRAM paralelným strojom, ktoré sú prakticky konštruovateľné? Nakoľko nerealistický je predpoklad neohraničeného počtu procesorov a pamäťových miest? Je jednotková cena inštrukcie adekvátna? Aby sme mohli dať odpoveď na všetky otázky, je vhodné uviesť ďalší paralelný výpočtový model, motivovaný pohľadom na spôsob, akým sa prevádzajú výpočty na najnižšej úrovni. Tento model sa ľahko popisuje a je vhodný pre podrobnejšiu matematickú analýzu.

Logické obvody modelujú elektrické obvody, realizujúce bitové operácie. Môžeme na ne preto nazerať ako na booleovské funkcie a zapisovať ich pomocou booleovských formúl. Avšak veľkosť booleovskej formuly neodráža dostatočne presne zložitosť booleovskej funkcie. Je to preto, že nezachytáva možnosť viacnásobného použitia čiastočného výsledku (napr. vo formule $(x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee x_2 \vee x_4 \vee \neg x_3) \wedge (x_1 \vee x_2 \vee \neg x_4)$ sa viackrát opakuje disjunkcia $x_1 \vee x_2$). Tento nedostatok odstraňuje nasledujúci, všeobecnejší formalizmus.

Nech G je orientovaný acyklický graf, ktorého vrcholy sú označené. Korene, t.j. vrcholy do ktorých nevchádzajú žiadne hrany, sa nazývajú *vstupné vrcholy*. Vrcholy, ktorých prichádzajúci stupeň je $d > 0$, nazývame *vnútorné*. Listy grafu, t.j. vrcholy, z ktorých nevychádzajú žiadne hrany, sa nazývajú *výstupné vrcholy*.

Každému vstupnému vrcholu priradíme literál (booleovskú premennú alebo jej negáciu). Každému vnútornému vrcholu v grafu priradíme booleovskú funkciu $F_v : \{0,1\}^d \to \{0,1\}$, kde d je prichádzajúci stupeň vrcholu v. Prichádzajúce hrany sú usporiadané a premenné funkcie F_v im zodpovedajú v tomto poradí. Takýto graf sa nazýva *logický obvod*.

Každý logický obvod C určuje jednu booleovskú funkciu nasledovne. Hodnotou vstupného vrcholu je hodnota literálu, ktorý mu je priradený. To je tzv. vstup výpočtu. Z neho vieme pre každý vrchol v vypočítať hodnotu $x(v) \in \{0,1\}$: ak počiatočné vrcholy $u_1, \dots, u_d$ hrán vstupujúcich do v už majú priradené hodnoty, tak vrcholu v priradíme hodnotu $F_v(x(u_1), \dots, x(u_d))$. Hodnota priradená výstupnému vrcholu je výstupom výpočtu. O funkcii, ktorá je takto definovaná hovoríme, že je počítaná obvodom C. Ak obvod má viac než jeden výstup, tak ho chápeme ako obvod, ktorá počíta súčasne viac funkcií, resp. ako obvod počítajúci vektorovú funkciu.



Obrázek 9.1: Booleovský obvod veľkosti 5 a hĺbky 3

Zrejme každá booleovská funkcia je počítaná triviálnym obvodom (hĺbky 1), ktorého jediný vnútorný vrchol počíta hodnotu funkcie priamo zo vstupu. Pojem logického obvodu sa stane zaujímavým v okamihu, keď ohraničíme množinu funkcií, ktoré môžeme priraďovať vnútorným vrcholom. Pre nás najzaujímavejším je prípad, keď všetky vnútorné vrcholy počítajú buď disjunkciu alebo konjunkciu dvoch premenných. Takýto obvod sa nazýva aj *boole-*

ovský obvod. Pretože konjunkcia, disjunkcia a negácia tvoria úplnú bázu, je možné ku každej booleovskej funkcii skonštruovať booleovský obvod, ktorý ju počíta. My budeme v ďaľšom uvažovať len booleovské obvody.

Dve najprirodzenejšie zložitostné miery obvodov sú jeho *veľkosť* a *hĺbka*. Zodpovedajú počtu procesorov a času potrebnému na výpočet funkcie. Pre daný obvod C definujeme jeho veľkosť, označujeme $SIZE(C)$, ako počet jeho vnútorných vrcholov. Jeho hĺbka, $DEPTH(C)$, je rovná dĺžke najdlhšej cesty zo vstupného do výstupného vrcholu. Pretože v booleovskom obvode vstupujú do každého vnútorného vrcholu dve hrany, je $SIZE(C) \leq 2^{DEPTH(C)}$.

Od zložitosti obvodu môžeme prejsť k zložitosti booleovskej funkcie f. Značíme ju $SIZE_f$ a definujeme ako

$$SIZE_f \stackrel{\text{def}}{=} \min \left\{ q \,\middle|\, \exists C : \; SIZE(C) = q \;\text{ a }\; C \text{ počíta } f \right\}$$

Nasledujúce tvrdenie ukazuje, že zložitosť booleovskej funkcie je zhora ohraničená funkciou exponenciálnou vzhľadom k počtu jej premenných.

Veta 9.7 *Nech f je booleovská funkcia n premenných, $f : \{0,1\}^n \to \{0,1\}$. Potom*

$$SIZE_f \leq 2^{n+1} - 3.$$

Dôkaz Každú booleovskú funkciu môžeme vyjadriť ako

$$f(x_1, \dots, x_n) = (x_1 \wedge f(1, x_2, \dots, x_n)) \vee (\neg x_1 \wedge f(0, x_2, \dots, x_n)).$$

Preto obvod pre funkciu n premenných sa dá vždy skonštruovať vhodným spojením dvoch obvodov pre funkcie $(n-1)$ premenných. Spojenie si vyžaduje nanajvýš dva $\wedge$ vrcholy a jeden $\vee$ vrchol, tj. spolu tri dodatočné vrcholy. Preto pre ľubovoľné $n \geq 1$ je $SIZE_f \leq g(n)$, kde

$$g(n) = \begin{cases} 1 & \text{ak } n = 1 \\ 2g(n-1) + 3 & \text{inak} \end{cases}$$

Ľahko overíme, že $g(n) \leq 2^{n+1} - 3$. □

Z druhej strany sa dá ukázať, že existuje booleovská funkcia, ktorej zložitosť sa približuje k hranici uvedenej v predošlej vete.

Veta 9.8 *Pre ľubovoľné dostatočne veľké n existuje booelovská funkcia n premenných taká, že*

$$SIZE_f > \frac{2^n}{n}.$$

Dôkaz V skutočnosti dokážeme ešte silnejšie tvrdenie, a to že skoro všetky booleovské funkcie n premenných majú zložitosť väčšiu než $\frac{2^n}{n}$. Dôkaz je založený na odhade počtu booleovských obvodov veľkosti s s $2n$ vstupnými vrcholmi. Označme toto číslo ako $C(n,s)$.

Najprv určíme horný odhad čísla $C(n,s)$ tak, že odhadneme počet obvodov, ktorých graf nie je nutne acyklický. Každý takýto obvod má $2n$ vstupných vrcholov a s vnútorných vrcholov s prichádzajúcim stupňom 2. Obidve vstupné hrany spájajú vnútorný vrchol s ľubovoľným z $2n + s$ vrchlov. Každému vnútornému vrcholu je priradená buď $\wedge$ alebo $\vee$. Jeden z vrcholov je výstupný. Spolu

$$C(n,s) \leq \frac{\overbrace{s}^{\text{výstup}} \cdot [\overbrace{2}^{\{\wedge,\vee\}} \cdot \overbrace{(2n+s)^2}^{\substack{\text{počet}\\ \text{možných}\\ \text{vstupov}}}]^{\overbrace{s}^{\substack{\text{počet}\\ \text{vnútorných}\\ \text{vrcholov}}}}}{\underbrace{s!}_{\text{očíslovanie nehrá úlohu}}}$$

Podľa Stirlingovej formuly je $s! \geq (\frac{s}{e})^s$, takže po zlogaritmovaní dostávame

$$\begin{aligned}\log C(n,s) &\leq s \log 2(2n+s)^2 + \log s - \log s! \\ &\leq 2s\log(2n+s) + s + \log s - s\log s + s\log e\,.\end{aligned}$$

Za predpokladu $s \geq n$ je

$$\begin{aligned}\log C(n,s) &\leq 2s\log 4s + s + \log s - s\log s + s\log e \\ &= (s+1)\log s + (5+\log e)s\,.\end{aligned}$$

Zvoľme $s = \frac{2^n}{n}$

$$\begin{aligned}\log C(n,s) &\leq (\frac{2^n}{n}+1)(n-\log n) + (5+\log e)\frac{2^n}{n} \\ &= 2^n - \frac{2^n\log n}{n}(1+o(1))\,.\end{aligned}$$

Ak sa vrátime naspäť k $C(n,s)$, tak vidíme, že počet funkcií vypočítateľných obvodom veľkosti nanajvýš $\frac{2^n}{n}$ je $2^{2^n} \cdot 2^{-\frac{2^n\log n}{n}(1+o(1))}$, čo je len veľmi malá časť spomedzi všetkých 2^{2^n} booleovských funkcií n premenných. □

Logické obvody (špeciálne booleovské obvody) sú dôležité z dvoch hľadísk. Jednak poskytujú kombinatorický popis niektorých jednoduchých elektrických sietí. Za druhé, a to je z nášho pohľadu dôležitejšie, poskytujú dobrý popis logickej štruktúry paralelného algoritmu. Jeho vrcholy zodpovedajú operáciam algoritmu. Usporiadanie vrcholov určuje poradie, v akom musia byť jednotlivé operácie prevádzané: operácia priradená počiatočnému vrcholu hrany sa musí vykonať skôr než operácia priradená koncovému vrcholu tejto hrany.

V ďalšom nás bude zaujímať, aký je vzťah booleovských obvodov a PRAMov, resp. iných výpočtových modelov. K jeho upresneniu potrebujeme prezentovať obvody vhodným spôsobom. Uvedieme jednu z možností.

Definícia 9.9 Štandartné kódovanie $\bar{C}$ *booleovského obvodu C je binárny reťazec pozostávajúci zo štvoríc $< v, g, l, r >$ (jedna štvorica pre každý vrchol obvodu), za ktorými nasledujú tri postupnosti čísiel vrcholov $< x_1, \ldots, x_n >, < \bar{x_1}, \ldots, \bar{x_n} >$ a $< y_1, \ldots, y_m >$. Predpokladáme, že vrcholy obvodu C sú ľubovoľne, ale fixne očíslované číslami 1,…, (počet vrcholov obvodu C). Štvorica $< v, g, l, r >$ opisuje vrchol v; g je funkcia priradená tomuto vrcholu, l resp. r je číslo ľavého resp. pravého vstupu (ak existujú). Postupnosť $< x_1, \ldots, x_n >$ ($< \bar{x_1}, \ldots, \bar{x_n} >$) popisuje vstupné vrcholy, ktorým sú priradené literály $x_1, \ldots, x_n$ ($\neg x_1, \ldots, \neg x_n$). Nakoniec postupnosť $< y_1, \ldots, y_m >$ popisuje výstupné vrcholy.*

Každú štvoricu môžeme zakódovať binárnym reťazcom dĺžky $\mathcal{O}(\log V)$, kde V je počet vrcholov obvodu C. Celkove je dĺžka štandartného kódu schémy C rovná $\mathcal{O}(V \log V)$.

Uniformné obvody

Konkrétny obvod s n vstupmi a m výstupmi je konečným objektom, počítajúcim funkciu z množiny binárnych reťazcov dĺžky n do množiny binárnych reťazcov dĺžky m. Dôsledkom je, že pre vstupy rôznej dĺžky potrebujeme rôzne obvody. To je ale v protiklade s našim doterajším chápaním výpočtov: jeden algoritmus bol schopný spracovávať vstupy ľubovoľnej dĺžky. Otázkou je, či sa dá nejakým spôsobom zobecniť pojem obvodov tak, aby sa priblížil takémuto chápaniu výpočtov.

Najjednoduchšia generalizácia je možná za predpokladu, že dĺžka výstupu m je funkciou (najlepšie konštantnou) len dĺžky vstupu. To znamená, že sa obmedzíme na jednoduché funkcie f, kde dĺžka reťazca $f(x)$ je rovnaká pre všetky x dĺžky n. Označme túto dĺžku ako $m(n)$. Potom funkciu $f : \{0,1\}^* \to \{0,1\}^*$ môžeme reprezentovať nekonečnou postupnosťou obvodov $\{C_n\}$, kde obvod C_n počíta funkciu f obmedzenú na vstupy dĺžky n.

Naopak, každej postupnosti obvodov $\{C_n\}$, kde každý obvod C_n počíta funkciu $f^n : \{0,1\}^n \to \{0,1\}^{m(n)}$ môžeme priradiť booleovskú funkciu $f_C : \{0,1\}^* \to \{0,1\}^*$ definovanú ako

$$f_C(x) = f^{|x|}(x).$$

Pre nás je zaujímavý špeciálny prípad, keď $m(n) = 1$ pre všetky n.

Definícia 9.10 *Nech* $C = \{C_n\}$ *je postupnosť booleovských obvodov počítajúcich booleovskú funkciu* $f_C : \{0,1\}^* \to \{0,1\}$. *Potom* jazyk akceptovaný postupnosťou $\{C_n\}$ *je definovaný ako*

$$L_C = \{x \in \{0,1\}^* \mid f_C(x) = 1\}.$$

Zložitostné miery definované pre konkrétny obvod môžeme prirodzeným spôsobom rozšíriť aj na postupnosť $C = \{C_n\}$ obvodov. Veľkosť $SIZE_C$ je definovaná ako funkcia na množine $\mathbb{N}$, kde $SIZE_C(n) = SIZE(C_n)$. Analogicky pre hĺbku. O jazyku L hovoríme, že má polynomiálny obvod práve ak existuje postupnosť obvodov C, ktorá akceptuje L a pritom $SIZE_C(n) \leq p(n)$ pre nejakú polynomiálnu funkciu p. Prirodzenou je otázka, aké typy jazykov majú polynomiálne obvody.

Lema 9.11 *Každý jazyk z* P *má polynomiálnu schému.*

Dôkaz Nech jazyk L je rozhodovaný deterministickým Turigovym strojom $\mathcal{M} = (Q, \Sigma, \Gamma, \triangleright, \sqcup, \delta, q_0)$ s časovou zložitosťou n^k. Bez straty na obecnosti môžeme predpokladať $L \subseteq \{0,1\}^*$, stroj $\mathcal{M}$ je jednopáskový a v akceptujúcej konfigurácii hlava sníma prvé políčko pásky. Výpočet stroja $\mathcal{M}$ na vstupe x môžeme potom prezentovať ako výpočtovú tabuľku T rozmerov $(|x|^k+1) \times (|x|^k+1)$. V tabuľke každý riadok zodpovedá jednému výpočtovému kroku (riadky číslujeme od 0 do $|x|^k$), zatiaľ čo stĺpce zodpovedajú pozíciam na páske stroja $\mathcal{M}$ (číslujeme ich rovnako ako riadky). Symbol $T_{i,j}$ na pozícii (i,j) teda reprezentuje obsah j–teho políčka v i–tej konfigurácii. Kvôli technickej jednoduchosti ďalej predpokladáme, že prázdnemu políčku stroja zodpovedá v tabuľke symbol $\sqcup$. Ak v i–tej konfigurácii je stav stroja q a pozícia hlavy s, tak na pozícii (i,s) nie je symbol σ zodpovedajúci obsahu s–teho políčka i–tej konfigurácii, ale nový symbol σ_q. Špeciálne ak i–ta konfigurácia je akceptujúca (zamietajúca), tak na $T_{i,1}$ je q_{accept} (q_{reject}). Nakoniec predpokladáme, že ak výpočet stroja $\mathcal{M}$ na vstupe x sa zastaví po $r < |x|^k$ krokoch, tak všetky riadky nasledujúce za r-tým sú s ním identické. Výpočtová tabuľka sa nazýva akceptujúca, práve ak $T_{|x|^k,1} = q_{accept}$. Existencia akceptujúcej výpočtovej tabuľky pre slovo x teda indikuje, že slovo je akceptované strojom $\mathcal{M}$.

Navrhnutá prezentácia výpočtov stroja $\mathcal{M}$ nám umožní priamočiaru konštrukciu postupnosti obvodov akceptujúcej jazyk L. Umožní nám to práve vlastnosť „lokálnosti" každého prvku tabuľky. Zrejme ak $i = 0$ alebo $j = 0$ alebo $j = |x|^k$ tak symbol $T_{i,j}$ je určený jednoznačne ako j-ty symbol z x alebo $\sqcup$ resp. $\triangleright$ resp. $\sqcup$. Uvažujme teraz ľubovoľný iný prvok $T_{i,j}$. Hodnota $T_{i,j}$ zodpovedá obsahu j–teho políčka v i–tej konfigurácii a *závisí len od obsahu toho istého políčka a s ním susediacich políčok v konfigurácii* $i-1$. Inak povedané, hodnota $T_{i,j}$ závisí len od hodnôt $T_{i-1,j-1}, T_{i-1,j}$ a $T_{i-1,j+1}$.

Nech Υ je abeceda symbolov, ktoré sa môžu vyskytovať v tabuľke T. Zakódujeme každý prvok $\gamma \in \Upsilon$ ako vektor $(s_1, \ldots, s_m)$, kde $s_1, \ldots, s_m \in \{0,1\}$ a $m = \lceil \log |\Upsilon| \rceil$. Na výpočtovú tabuľku tak môžeme nazerať ako na tabuľku obsahujúcu binárne symbolu S_{ijl}, kde $0 \leq i \leq n^k$, $0 \leq j \leq n^k$ a $1 \leq l \leq m$. Vieme už, že hodnota každého S_{ijl} závisí len od $3m$ hodnôt $S_{i-1,j-1,l'}, S_{i-1,j,l'}$ a $S_{i-1,j+1,l'}$, kde $1 \leq l' \leq m$, To znamená, že existuje m booleovských funkcií $F_1, \ldots, F_m$, každá z nich má $3m$ premenných a pre všetky $i, j > 0$

$$S_{ijl} = F_l(S_{i-1,j-1,1}, \ldots, S_{i-1,j-1,m}, S_{i-1,j,1}, \ldots, S_{i-1,j+1,m}).$$

Keďže každú booleovskú funkciu vieme počítať obvodom, tak existuje booleovský obvod s $3m$ vstupmi a m výstupmi, ktorý pre vstupy $T_{i-1,j-1}, T_{i-1,j}$ a $T_{i-1,j+1}$ vypočíta binárne kód hodnoty T_{ij}. Tvar obvodu C závisí len od $\mathcal{M}$ a má fixovanú, konštantnú veľkosť, nezávislú od dĺžky x.

Teraz už ľahko navrhneme obvod C_n, ktorý pre každý reťazec x dĺžky n vypočíta hodnotu true vtedy a len vtedy, ak $x \in L$. Obvod C_n pozostáva z $(|x|^k)(|x|^k - 1)$ kópií obvodu

C, jedna kópia pre každé T_{ij} výpočtovej tabuľky s výnimkou prvého riadku a krajných stĺpcov. Označme C_{ij} (i,j)-tu kópiu C. Pre $i \geq 1$ vstupné vrcholy z C_{ij} stotožníme s výstupnými vrcholmi z $C_{i-1,j-1}$, $C_{i-1,j}$ a $C_{i-1,j+1}$. Vstupné vrcholy celého obvodu sú vrcholy zodpovedajúce vrcholom prvého riadku, prvého a posledného stĺpca. Výstupným vrcholom obvodu je prvý výstup obvodu $C_{|x|^k}$ za predpokladu, že prvý bit reťazca kódujúceho q_{accept} je 1 a prvý bit reťazca kódujúceho q_{reject} je 0.

Ľahko sa presvedčíme že postupnosť C_n akceptuje práve jazyk L. Jej veľkosť je $\mathcal{O}(n^{2k})$ a hĺbka $\mathcal{O}(n^k)$. □

Opačná implikácia vôbec nie je pravdivá:

Lema 9.12 *Existuje jazyk, ktorý nie je rekurzívny a pritom má obvod polynomiálnej veľkosti.*

Dôkaz Nech L je ľubovoľný nerozhodnuteľný jazyk nad abecedou $\{0,1\}$ a nech $U \subseteq \{1\}^*$ je jazyk $U = \left\{ 1^n \mid \text{binárny zápis čísla } n \text{ patrí do } L \right\}$. U je zrejme unárny, nerozhodnuteľný jazyk. Pritom preň existuje veľmi jednoduchá postupnosť obvodov $\{C_0, C_1, \dots\}$. Ak $1^n \in U$, tak C_n pozostáva z $n-1$ vrcholov typu $\wedge$, ktoré vypočítajú konjunkciu všetkých vstupov.Výstup je true práve ak vstupom je 1^n. Ak $1^n \notin U$, tak C_n má jediný výpočtový vrchol, ktorý je zároveň výstupným a počíta konštantu false. □

Predchádzajúca lema naznačuje, prečo nie sú obvody realistickým výpočtovým modelom: umožňujú investovať neohraničené množstvo výpočtu do konštrukcie každého obvodu z postupnosti (čoho dôsledkom je napr. existencia polynomiálneho obvodu pre nerozhodnuteľný jazyk), čo je evidentne neakceptovateľné. Obvody tohoto typu sa nazývajú *neuniformné*. Neuniformné obvody hrajú významnú úlohu pri získavaní dolných odhadov zložitosti. Naopak, v situácii keď potrebujeme odhadnúť zložitosť konkrétneho problému zhora sú nerealistické—neexistuje pre ne efektívny spôsob, ako pre dané n získať popis n–tej schémy C_n.

Jeden zo spôsobov, ako sa vyhnúť takýmto problémom je požadovať existenciu algoritmu, generujúceho jednotlivé členy postupnosti. Postupnosť obvodov je potom jednoznačne definovaná prostredníctvom algoritmu (výpočtového zariadenia), ktorý pre vstup n poskytne ako výstup štandartný kód obvodu C_n. Hovoríme o *uniformných* obvodoch. Samozrejme je veľmi dôležité, aká je zložitosť algoritmu generujúceho kódy obvodov. Intuitívne, konštruktor obvodov by nemal byť výpočtovo silnejší než obvody, ktoré generuje. Jedna z možností je chápať konštruktor ako Turingov stroj s logaritmickou priestorovou zložitosťou.

Definícia 9.13 (Uniformné obvody) *Postupnosť $\{C_n\}$ booleovských obvodov sa nazýva* logaritmicky priestorovo uniformná *práve ak transformácia $1^n \to \bar{C}_n$ je vypočítateľná deterministickým Turingovym strojom s priestorovou zložitosťou $\mathcal{O}(\log V_n)$, kde V_n je veľkosť obvodu C_n.*

Poznamenávame, že zložitosť algoritmu počítajúceho kódy obvodov je vyjadrená ako funkcia veľkosti vypočítaného kódu. V prípade, že pracujeme so schémami polynomiálnej veľkosti a vstupný parameter n algoritmu zadávame unárne, tak zložitosť algoritmu je logaritmická aj ako funkcia dĺžky vstupu.

Logaritmická priestorová uniformnosť sa niekedy nazýva aj Borodin—Cookova uniformita, podľa autorov, ktorí ju ako prví definovali. Výhodou tohoto prístupu je, že kód schémy $\bar{C}_n$ sa dá vypočítať sekvenčne v polynomiálnomom čase a paralelne v polylogaritmickom čase s polynomiálnym počtom procesorov, a teda je vhodný ako pre sekvenčné, tak aj paralelné počítanie.

Uvedený spôsob chápania uniformity samozrejme nie je jediný možný. Tak, ako vládne nejednotnosť v tom, ktorý paralelný výpočtový model je najlepší, tak aj v prípade uniformity existuje mnoho rôznych prístupov. Rôzne prístupy sú užitočné v rôznych situáciách. Napríklad, ak upustíme od požiadavku, aby sa kódy dali počítať efektívne paralelnými algoritmami, je vhodnejšie pracovať s tzv. *P–uniformitou* (požadujeme, aby kód obvodu bol vypočítateľný deterministickým Turingovym strojom v polynomiálnom čase). Pojem logaritmickej priestorovej uniformity nie je vhodný, ak pracujeme s obvodmi malej hĺbky.

V ďaľšom budeme pod uniformnými obvodmi rozumieť vždy logaritmicky priestorovo uniformné obvody.

Veta 9.14 *Jazyk L má uniformnú polynomiálnu schému práve ak $L \in \mathsf{P}$.*

Dôkaz Jednu implikáciu sme dokázali v leme 9.11: stačí overiť, že naznačená konštrukcia sa dá realizovať v logaritmickom priestore.

Z druhej strany predpokladajme, že L má uniformný polynomiálny obvod. Potom o slove x vieme rozhodnúť, či patrí do L tak, že skonštruujeme v logaritmickom priestore (a teda aj polynomiálnom čase) obvod $C_{|x|}$ a v polynomiálnom čase vyhodnotíme jeho výstup pre vstup x. □

Vzájomné simulácie

Pre lepšie pochopenie vzťahu medzi obidvoma definovanými paralelnými výpočtovými modelmi uvádzame, ako sa dajú navzájom simulovať. Prvý výsledok hovorí o simulácii uniformnej postupnosti schém PRAMom.

Lema 9.15 *Ak funkcia F je počítaná uniformnou postupnosťou obvodov $C = \{C_n\}$ s veľkosťou $SIZE_C(n) = s(n)$ a hĺbkou $DEPTH_C(n) = d(n)$, tak existuje PRAM počítajúci funkciu F v čase $\mathcal{O}(d(n))$ a s $\mathcal{O}(s(n))$ procesormi.*

Dôkaz Stroj PRAM pre vstup dĺžky n najprv vypočíta kód obvodu C_n tak, že simuluje Turingov stroj generujúci kód obvodu C_n (C je uniformná). Každému vrcholu g_i obvodu bude zodpovedať zdieľaný pamäťový register c_i a každému vnútornému vrcholu jeden procesor P_i. Program stroja P_i je veľmi jednoduchý: najprv čaká $4d$ krokov, kde d je dĺžka najdlhšej cesty z nejakého vstupného vrchola do vrcholu g_i. Potom P_i vypočíta hodnotu priradenú vrcholu g_i a uloží ju do spoločného registra c_i. Ak predpokladáme, že výstupné vrcholy sú $g_1, \dots, g_m$, tak výsledok bude na konci výpočtu uložený v registroch $c_1, \dots, c_m$. □

Simulácia strojov PRAM pomocou obvodov je technicky náročnejšia; hlavná myšlienka spočíva, podobne ako v dôkaze lemy 9.11, v zápise výpočtu stroja PRAM ako postupnosti obvodov a jeho následnom prevedení do obvodu.

Lema 9.16 *Ak funkcia F je počítaná PRAMom v čase $f(n)$ s $g(n)$ procesormi (f, g sú konštruovateľné funkcie), tak existuje uniformná postupnosť obvodov hĺbky $\mathcal{O}(f(n)(f(n) + \log n))$ a veľkosti $\mathcal{O}(g(n)f(n)(n^k f(n) + g(n))$ počítajúca F, pričom n^k je časová zložitosť Turingovho stroja generujúceho program pre stroj PRAM.*

Spojením uvedených simulácií dostávame špeciálne pre polylogaritmickú časovú zložitosť a polynomiálny počet procesorov vzťah:

Veta 9.17 *Funkcia $f : \{0,1\}^* \to \{0,1\}^*$ sa dá počítať uniformnou postupnosťou booleovských obvodov C s hĺbkou $DEPTH_C = (\log n)^{\mathcal{O}(1)}$ a veľkosťou $SIZE_C = n^{\mathcal{O}(1)}$ vtedy a len vtedy, ak f sa dá počítať CREW–PRAM strojom v paralelnom čase $t(n) = (\log n)^{\mathcal{O}(1)}$ za použitia $p(n) = n^{\mathcal{O}(1)}$ procesorov.*

Výsledok ukazuje, že hĺbka obvodu zodpovedá paralelnému času a veľkosť obvodu počtu procesorov.

9.2.3 Paralelná výpočtová téza

Predpokladajme, že máme zložitostnú triedu definovanú ako triedu jazykov rozhodovaných časovo ohraničenými paralelnými strojmi. Je možné charakterizovať takúto triedu jazykov aj prostredníctvom deterministických sekvenčných časovo ohraničených strojov, konkrétne Turingovych strojov? Štúdium viacerých paralelných výpočtových modelov ukázalo, že ak rozumným spôsobom ohraničíme počet a silu paralelných procesorov, tak platí obecné pravidlo, nazývané *Paralelná výpočtová téza*.

Označme $\mathcal{P}$ paralelný výpočtový model. Pre model $\mathcal{P}$ platí paralelná výpočtová téza práve ak existujú dva polynómy p a q také, že

$$\mathcal{P} - \mathsf{TIME}(f(n)) \subseteq \mathsf{DSPACE}(p(f(n))) \text{ a } \mathsf{DSPACE}(g(n)) \subseteq \mathcal{P} - \mathsf{TIME}(q(g(n))).$$

Pritom $\mathcal{P} - \mathsf{TIME}(f(n))$ označuje triedu jazykov rozhodovaných paralelnými strojmi typu $\mathcal{P}$ v čase $\mathcal{O}(f(n))$. Inak formulované,

> *Paralelný výpočtový model $\mathcal{P}$ spĺňa paralelnú výpočtovú tézu práve ak trieda jazykov rozhodovaných strojmi typu $\mathcal{P}$ v polynomiálnom čase je zhodná s triedou jazykov rozhodovaných v sekvenčnom polynomiálnom priestore (tj. s triedou* PSPACE*).*

Hoci nie všetky paralelné výpočtové modely vyhovujú paralelnej výpočtovej téze, nami uvažované modely PRAM strojov a uniformných booleovských obvodov jej vyhovujú. To potvrdzuje korektnosť nášho výberu paralelných výpočtových modelov.

Veta 9.18 *Stroje PRAM a uniformné obvody spĺňajú paralelnú výpočtovú tézu.*

Dôkaz Vzhľadom na tvrdenie Vety 9.17 stačí ukázať platnosť paralelnej výpočtovej tézy len pre jeden model; naznačíme to pre uniformné obvody.

Prvá časť tézy: Uvažujme obvod C_n veľkosti $s(n)$ a hĺbky $d(n)$. Potom sa dá skonštruovať deterministický Turingov stroj, ktorý ak dostane kód obvodu C_n spolu s jeho vstupom, tak v priestore $\mathcal{O}(d(n))$ vypočíta výstup obvodu C_n. Turingov stroj vyhodnocuje schému počínajúc od výstupného vrcholu a postupujúc naspäť k vstupným vrcholom. Pritom si v zásobníku uchováva údaje o prejdenej ceste: keďže hĺbka schémy je $d(n)$, tak veľkosť zásobníku je tiež $d(n)$ a každý údaj zásobníku je konštantnej veľkosti. Preto stačí Turingovmu stroju na uloženie zásobníka priestor $\mathcal{O}(d(n))$. To, spolu s unifomitou postupnosti obvodov, potvrdzuje platnosť prvej podmienky paralelnej výpočtovej tézy.

Druhá časť tézy: Máme ukázať, aký je vzťah medzi sekvenčným priestorom a paralelným časom. Nech T je $g(n)$–priestorovo ohraničený Turingov stroj a g je konštruovateľná polynomiálna funkcia. Počet rôznych konfigurácií stroja T vo výpočte na vstupe w dĺžky n je nanajvýš $m = 2^{cg(n)}$, kde c je nejaká konštanta. Nech S_1 označuje počiatočnú konfiguráciu. Naviac môžeme predpokladať existenciu jedinej akceptujúcej konfigurácie, S_m.

Uvažujme nasledovný paralelný algoritmus: Najprv sa skonštruuje $m \times m$ prechodová matica A; položka $A(i,j)$ je rovná 1 práve ak stroj T môže v nanajvýš jednom kroku výpočtu prejsť z konfigurácie S_i do konfigurácie S_j, inak $A(i,j) = 0$. V ďalšom sa vypočíta tranzitívny uzáver matice A, t.j. matica $B = A^m$. Ľahko overíme, že T akceptuje vstup w práve ak $B(1,m) = 1$, to znamená, ak konfigurácia S_m je dosiahnuteľná z konfigurácie S_1. Pritom matica A sa dá vypočítať uniformným obvodom v $\mathcal{O}(\log m)$ krokoch. Tranzitívny uzáver matice A sa dá vypočítať v $\mathcal{O}((\log m)^2)$ krokoch, teda celkove v $\mathcal{O}((g(n)^2)$ paralelných krokoch. □

Paralelné výpočtové modely spĺňajúce paralelnú výpočtovú tézu tvoria tzv. *druhú počítačovú triedu*. Trieda jazykov rozhodovaných výpočtovým modelom z druhej počítačovej triedy v polynomiálnom čase je teda práve PSPACE. Hoci intuitívne sa zdá, že trieda PSPACE je oveľa bohatšia ako trieda P (teda trieda jazykov rozhodovaných výpočtovým modelom z prvej počítačovej triedy v polynomiálnom čase), žiaden dôkaz tohoto tvrdenia nie je známy.

9.3 Trieda NC

Našim cieľom je nájsť exaktný popis triedy všetkých vysoko paralelizovateľných problémov a následne skúmať vzťah tejto triedy a triedy P. Pretože trieda P je definovaná ako trieda formálnych jazykov s istými vlastnosťami a na druhej strane máme paralený RAM, počítajúci funkcie, potrebujeme transformáciu, ktorá nám umožní oba pohľady zjednotiť.

Definícia 9.19 *Nech L je jazyk nad abecedou $\{0,1\}$.* Charakteristická funkcia *jazyka L je funkcia f_L definovaná pre všetky $x \in \{0,1\}^*$ takto: $f_L(x) = 1$ ak $x \in L$ a $f_L(x) = 0$ ak $x \notin L$.*
Jazyk L je rozhodovaný v paralelnom čase $t(n)$ s $p(n)$ procesormi práve ak jeho charakteristická funkcia f_L je počítaná v paralelnom čase $t(n)$ s $p(n)$ procesormi.

Ak vezmeme do úvahy, že každý sekvenčný proces bežiaci v polynomiálnom čase sa dá simulovať polynomiálnym počtom procesorov bežiacich v polynomiálnom čase, a naopak, dostávame nasledovný vzťah:

Lema 9.20 *Jazyk L je rozhodovaný v sekvenčnom čase $n^{\mathcal{O}(1)}$ vtedy a len vtedy, ak L je rozhodovaný v paralelnom čase $n^{\mathcal{O}(1)}$ s $n^{\mathcal{O}(1)}$ procesormi.*

Inak povedané, trieda paralelne riešiteľných problémov je zhodná s triedou P. Definujeme triedu vysoko paralelizovateľných problémov NC.

Definícia 9.21 (Trieda NC) *Trieda NC je trieda jazykov, ktoré sú rozhodované v paralenom čase $(\log n)^{\mathcal{O}(1)}$ s $n^{\mathcal{O}(1)}$ procesormi.*

Vzťah medzi triedami P a NC ukazuje nasledujúca Lema:

Lema 9.22 $\mathsf{NC} \subseteq \mathsf{P}$

Dôkaz Je analogický s prvou časťou dôkazu vety 9.18. Pretože v tomto prípade je hĺbka postupnosti schém polylogaritmická, je časová zložitosť Turingovho stroja, ktorý ju simuluje, polynomiálna. □

> *Základnou otázkou paralených výpočtov je, či táto inklúzia je ostrá*, t.j. či existujú vnútorne sekvenčné problémy.

Ak sa na triedu NC pozrieme prostredníctvom booleovských obvodov, získame jemnejšie členenie:

Definícia 9.23 (Trieda NC^k) *Pre každé celé číslo $k \geq 1$, je trieda NC^k definovaná ako trieda všetkých jazykov L takých, že charakteristická funkcia f_L jazyka L je počítaná uniformnou postupnosťou booleovských obvodov C so zložitosťou $DEPTH_C = (\log n)^k$ a $SIZE_C = n^{\mathcal{O}(1)}$.*

Poznamenávame, že $\mathsf{NC} = \bigcup_{k \geq 1} \mathsf{NC}^k$. Z podrobnej analýzy dôkazu Vety 9.18 plynie vzťah

$$\mathsf{NC}^1 \subseteq \mathsf{DLOG} \subseteq \mathsf{NLOG} \subseteq \mathsf{NC}^2$$

Pre ilustráciu uvádzame niekoľko príkladov problémov patriacich do triedy NC^1 resp. NC^2.
Trieda NC^1

1. Sčítanie dvoch binárnych čísiel dĺžky m.
2. Násobenie dvoch binárnych čísiel dĺžky m.
3. Delenie dvoch binárnych čísiel dĺžky m.
4. Suma m binárnych čísiel dĺžky m.
5. Súčin dvoch $m \times m$ matíc obsahujúcich binárne čísla dĺžky l.

Trieda NC^2

1. Determinant $m \times m$ matice obsahujúcej binárne čísla dĺžky l.
2. Inverzia $m \times m$ matice obsahujúcej binárne čísla dĺžky l.
3. Triedenie m binárnych čísiel dĺžky l.

9.3.1 P–úplnosť

Pri skúmaní vzťahu medzi triedami P a NP sa ukázal ako veľmi užitočný koncept NP–úplnosti, ktorý nám umožňoval klasifikovať relatívne najťažšie problémy z triedy NP a teda aj vhodných kandidátov na rozdiel $\mathsf{NP} \setminus \mathsf{P}$. Podobne je tomu aj pri triedach P a NC. Ovšem vzhľadom na to, že sa pohybujeme v triede problémov vypočítateľných paralelne v polylogaritmickom čase, je redukovateľnosť vypočítateľná v polynomiálnom čase zjavne príliš hrubým nástrojom a potrebujeme ho preto zjemniť. Aby sme mohli definovať vhodný typ redukcie, potrebujeme najprv poznať funkcie vypočítateľné paralelne v polylogaritmickom čase a s polynomiálnym počtom procesorov.

Definícia 9.24 (Trieda FNC) *Trieda* FNC *je trieda funkcií vypočítateľných v paralenom čase* $(\log n)^{\mathcal{O}(1)}$ *s* $n^{\mathcal{O}(1)}$ *procesormi.*

Podobným spôsobom, ako sa trieda jazykov NC delí na podtriedy NC^k sa aj trieda funkcií FNC delí na podtriedy FNC^k. Pritom definícia oboch nových tried je robustná v zmysle uzavretosti na kompozíciu dvoch funkcií.

Definícia 9.25 (NC–redukovateľnosť) *Jazyk* L_1 *je* NC–redukovateľný *na jazyk* L_2*, značíme* $L_1 \leq_{\mathsf{NC}} L_2$*, práve ak existuje funkcia* $R \in \mathsf{FNC}$ *taká, že* $w \in L_1$ *práve vtedy ak* $R(w) \in L_2$.
Jazyk L_1 *je* NC^k–redukovateľný *na jazyk* L_2, $L_1 \leq_{\mathsf{NC}^k} L_2$*, práve ak redukcia* R *je z* FNC^k.

Podobne ako (polynomiálna) redukcia, aj NC a NC^k redukovateľnosti majú dôležitú vlastnosť tranzitivity, tj. ak R je NC (NC^k) redukcia jazyka L_1 na jazyk L_2 a R' je NC (NC^k) redukcia jazyka L_2 na jazyk L_3, tak kompozícia $R \cdot R'$ je NC (NC^k) redukciou L_1 na L_3. Tranzitivita definovaných redukií nám umožňuje uvažovať o relatívne najťažšom probléme z danej zložitostnej triedy — tzv. úplnom probléme.

Definícia 9.26 (Úplnosť) *Jazyk* L *je* P–ťažký vzhľadom na NC–redukovateľnosť *práve ak* $L' \leq_{\mathsf{NC}} L$ *pre každé* $L' \in \mathsf{P}$.
Ak naviac jazyk $L \in \mathsf{P}$*, tak sa nazýva* P–úplný vzhľadom na NC–redukovateľnosť.

Pretože triedy P a NC sú uzavreté vzhľadom na NC a NC^k–redukovateľnosť (porovnaj s uzavretosťou tried P, NP, PSPACE vzhľadom na polynomiálnu redukovateľnosť), môžeme formulovať nasledujúci vzťah

Veta 9.27 *Ak jazyk, ktorý je P–úplný vzhľadom na NC–redukovateľnosť patrí do triedy NC, tak* $\mathsf{P} = \mathsf{NC}$.

Inak vyjadrené: $\mathsf{P} \neq \mathsf{NC}$ implikuje, že žiaden P–úplný problém nepatrí do NC. Preto P–úplné problémy sú najvážnejšími kandidátmi na vnútorne sekvenčné problémy, t.j. problémy, ktoré sa nedajú riešiť paralelne v polylogaritmickom čase s polynomiálnym počtom procesorov.

Zatiaľ sme však o žiadnom konkrétnom probléme nedokázali, že je P–úplný. V nasledujúcej časti to ukážeme pre problémy CVP a MAXIMÁLNY TOK.

P–úplnosť problému CVP

Problém hodnoty booleovského obvodu, skrátene CVP (z anglického Circuit Value Problem), je formulovaný nasledovne: je daný štandartný kód $\bar{C}$ booleovského obvodu C počítajúceho funkciu $f : \{0,1\}^n \to \{0,1\}$ a je daný vektor hodnôt premenných $x \in \{0,1\}^n$. Úlohou je určiť výstup obvodu, t.j. hodnotu $f(x)$.

Veta 9.28 *Problém CVP je P–úplný vzhľadom na NC–redukovateľnosť.*

Dôkaz Príslušnosť problému CVP do triedy P sa overí jednoducho. Ukážeme, že každý jazyk $L \in \mathsf{P}$ je NC–redukovateľný na problém CVP. Znamená to, že musíme nájsť redukciu, ktorá každému slovu w priradí prípad $(\bar{C}, x)$ problému CVP (t.j. kód obvodu a jeho vstup) tak, že $w \in L$ práve ak výstup obvodu C pre vstup x je 1.

Predpokladajme $L \subseteq \{0,1\}^*$. Z $L \in \mathsf{P}$ plynie existencia jednopáskového determinisitického Turingovho stroja $\mathcal{M} = (Q, \Sigma, \Gamma, \triangleright, \sqcup, \delta, q_1)$ rozhodujúceho jazyk L v polynomiálnom čase $T(n)$. Nech $Q = (q_1, \ldots, q_s)$, $\Gamma = (a_1, \ldots, a_m)$. Ďalej bez straty všeobecnosti môžememe predpokladať že v akceptujúcej konfigurácii je na prvom políčku zapísaná 1. Naznačíme, ako sa dá skonštruovať obvod C a jeho vstup tak, aby výstup obvodu bol 1 práve ak $\mathcal{M}$ akceptoval.

Obvod C počíta konjunkciu nasledujúcich booleovských funkcií:

$H(i,t)$ $H(i,t) \stackrel{\text{def}}{=} 1$ práve ak ak v t–tom kroku výpočtu sníma hlava stroja $\mathcal{M}$ i–te políčko; $0 \leq i \leq T(n)$, $0 \leq t \leq T(n)$.

$C(i,j,t)$ $C(i,j,t) \stackrel{\text{def}}{=} 1$ práve ak v t–tom kroku výpočtu stroja $\mathcal{M}$ je obsah i–teho políčka a_j; $0 \leq i \leq T(n)$, $1 \leq j \leq m$, $0 \leq t \leq T(n)$.

$S(k,t)$ $S(k,t) \stackrel{\text{def}}{=} 1$ práve ak v t–tom kroku výpočtu je stroj $\mathcal{M}$ v stave q_k; $1 \leq k \leq s$, $0 \leq t \leq T(n)$.

Obvod bude pozostávať z $T(n)$ úrovní; t–tá úroveň obsahuje vrcholy počítajúce funkcie $H(i,t)$, $C(i,j,t)$ a $S(k,t)$, pre všetky $0 \leq i \leq T(n)$, $1 \leq j \leq m$ a $1 \leq k \leq s$. Počet vrcholov na jednej úrovni je $\mathcal{O}(T(n))$, pretože s a m sú konštanty závisiace len od stroja $\mathcal{M}$ a nie od jeho vstupu. Najprv špecifikujeme vrcholy nultej úrovne, ktoré sú vstupnými vrcholmi obvodu. Potom ukážeme, ako sa dajú vyjadriť $H(i,t+1)$, $C(i,j,t+1)$ a $S(k,t+1)$ pomocou $H(*,t)$, $C(*,*,t)$ a $S(*,t)$. Tak získame presný popis obvodu C.

Pre $t = 0$, hlava stroja $\mathcal{M}$ sníma prvé políčko, vstup $x = (x_1, \ldots x_n)$ je zapísaný na prvých n políčkach a $\mathcal{M}$ je v stave q_1. Preto do jednotlivých vstupných vrcholov vstupujú hodnoty takto:

$$H(i,0) = \begin{cases} 1 & \text{ak } i = 1; \\ 0 & \text{inak.} \end{cases}$$

$$C(i,j,0) = \begin{cases} 1 & \text{ak } i\text{–te políčko obsahuje } a_j; \\ 0 & \text{inak.} \end{cases}$$

$$S(k,0) = \begin{cases} 1 & \text{ak } k = 1; \\ 0 & \text{inak.} \end{cases}$$

Ďalej budeme postupne špecifikovať $H(i,t+1)$, $C(i,j,t+1)$ a $S(k,t+1)$.

Nech $I_R = \{(k,j) \mid \delta(q_k,a_j) = (*,*,1)\}$ a $I_L = \{(k,j) \mid \delta(q_k,a_j) = (*,*,-1)\}$. To znamená, I_R (I_L) obsahuje všetky také dvojice (q_k,a_j), ktoré spôsobia, že hlava stroja $\mathcal{M}$ sa posunie doprava (doľava). Potom $H(i,t+1)$ sa dá vyjadriť ako

$$\begin{aligned} H(i,t+1) =& (H(i-1,t) \wedge \bigvee_{(k,j)\in I_R} C(i-1,j,t) \wedge S(k,t)) \\ & \vee (H(i+1,t) \wedge \bigvee_{(k,j)\in I_L} C(i+1,j,t) \wedge S(k,t)). \end{aligned}$$

Z tohoto vyjadrenia vidno, ako sa $H(i,t+1)$ dá vyjadriť ako postupnosť AND a OR vrcholov dĺžky $\mathcal{O}(|Q| \cdot |\Sigma| = sm)$, čo je konštanta. Naviac, všetky $H(i,t)$ sa dajú vygenerovať v čase $\mathcal{O}(1)$ s použitím $\mathcal{O}(T^2(n))$ procesorov.

Uvažujme teraz $C(i,j,t+1)$. Pripomíname, že $C(i,j,t+1)$ vyjadruje, fakt, že obsah i–teho políčka v čase $t+1$ je a_j. Preto $C(i,j,t+1) = 1$ buď ak obsah i–teho políčka je a_j a v t–tom kroku výpočtu hlava nie je nastavená na i–tom políčku, alebo ak v $t+1$–vom kroku hlava zapíše a_j na i–te políčko. Preto

$$C(i,j,t+1) = \neg H(i,t) \wedge C(i,j,t) \vee H(i,t) \wedge (\bigvee_{\{(k,j') \mid \delta(q_k,a_{j'})=(*,a_j,*)\}} C(i,j',t) \wedge S(k,t)).$$

Podobne ako v prípade funkcie $H(i,t)$ aj funkcia $C(i,j,t)$ sa dá vyjadriť postupnosťou AND a OR vrcholov konštantnej dĺžky. Opäť, všetky $C(i,j,t)$ sa dajú vygenerovať v čase $\mathcal{O}(1)$ s použitím $\mathcal{O}(T^2(n))$ procesorov.

Nakoniec uvažujme $S(k,t+1)$. Nech $I_k = \{(k',j) \mid \delta(q_{k'},a_j) = (q_k,*,*)\}$. Potom $S(k,t+1)$ sa dá vyjadriť ako

$$S(k,t+1) = \bigvee_{1\leq i\leq T(n),(k',j)\in I_k} S(k',t) \wedge H(i,t) \wedge C(i,j,t).$$

Teda $S(k,t)$ sa dá vyjadriť ako postupnosť AND a OR vrcholov dĺžky $\mathcal{O}(T(n))$. Takúto postupnosť dokážeme vygenerovať v čase $\mathcal{O}(\log n)$ použitím $\mathcal{O}(T^2(n))$ procesorov.

Výstupným vrcholom obvodu je $C(1,r,T(n))$, kde $a_r = 1$. Z predchádzajúceho popisu je zrejmé, že celý obvod C sa dá vygenerovať NC–algoritmom, lebo $T(n)$ je polynomiálna funkcia. Výstup obvodu je 1 práve ak $\mathcal{M}$ akceptoval svoj vstup. Tým sme ukázali, že CVP je P–úplný problém. □

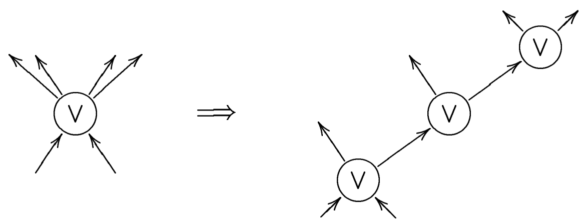
Ľahko sa ukáže, že aj nasledujúce modifikácie problému CVP sú P–úplné:

◇ **MCVP**: Problém hodnoty motónneho obvodu[2]

◇ **MCVP2**: Problém hodnoty motónneho obvodu, v ktorom odchádzajúci stupeň[3] každého vnútorného a vstupného vrcholu je nanajvýš 2 a výstupný vrchol počíta funkciu OR.

P–úplnosť problému MCVP sa ukáže jendoducho: jediné miesto, kde sme v predchádzajúcom dôkaze použili negáciu bolo pri vyjadrovaní funkcie $C(i, j, t + 1)$. Ale $\neg H(i, t) = \bigvee_{\substack{1 \le l \le T(n) \\ l \ne i}} H(l, t)$, lebo hlava musí snímať nejaké políčko v čase t.

K dôkazu P–úplnosti problému MCVP2 potrebujem ukázať, ako sa dá znížiť odchádzajúci stupeň vrcholu. Táto jednoduchá konštrukcia je naznačená na obrázku 9.2:



Obrázek 9.2: Redukcia odchádzajúceho stupňa vrcholu

P–úplnosť problému MAXIMÁLNY TOK

Tento problém[4] sme skúmali už v príklade 9.4, kde sme uviedli, že preň nie je známy žiaden vysoko paralelný algoritmus. Ukážeme, že sa jedná o P–úplný problém a teda existencia takéhoto algoritmu pre problém MAXIMÁLNY TOK by v skutočnosti implikovala $\mathsf{P} = \mathsf{NC}$.

Nech G je orientovaný graf $G = (V, H)$, s (t) je vrchol v G s nulovým prichádzajúcim (odchádzajúcim) stupňom a $c : H \to \mathbb{N}$ je funkcia, ktorá každej hrane priradí jej kapacitu. Nech f je tok v G ($f : H \to \mathbb{N}$, $f(i, j) \le c(i, j)$ pre $(i, j) \in H$). Potom *rezervnou cestou pre tok f* v grafe G je neorientovaná cesta Q z s do t taká, že pre každé dva susedné vrcholy i, j cesty Q platí:

1. $(i, j) \in H$ implikuje $f(i, j) < c(i, j)$. Takúto hranu nazývame súhlasnou.
2. $(j, i) \in H$ implikuje $f(j, i) > 0$. Takúto hranu nazývame spätnou.

Tok f je maximálny práve ak preň neexistuje žiadna rezervná cesta. Tento fakt využijeme pri dôkaze P–úplnosti problému MAXIMÁLNY TOK.

Veta 9.29 *Problém* MAXIMÁLNY TOK *je* P*–úplný vzhľadom na* NC*–redukovateľnosť.*

Dôkaz Navrhneme redukciu problému MCVP2 na problém MAXIMÁLNY TOK.

Nech obvod C a vstup x sú prípadom problému MCVP2. Skonštruujeme graf $G = (V, H)$ taký, že maximálny tok v G je nepárny práve ak výstup obvodu je 1. Naviac, G sa bude dať skonštruovať NC–algoritmom. Aby bol popis grafu G jednoduchší, zavedieme nasledovnú notáciu: $g_0, \ldots, g_n$ sú vnútorné vrcholy obvodu C; g_0 je výstupný vrchol. Štruktúra grafu G bude skoro totožná s obvodom — acyklickým grafom G; G má naviac dva vrcholy s a t a hrany s nimi incidujúce.

Presnejšie $V = \{0, 1, \ldots, n\} \cup \{s, t\}$. Vrchol i zodpovedá vrcholu g_i obvodu C. Množina hrán H obsahuje (pozri obrázok 9.3):

---

[2]monotónny obvod je booleovský obvod, v ktorom vstupným vrcholom priraďujeme len booleovské premenné (a nie ich negácie).

[3]počet hrán vychádzajúcich z vrchola

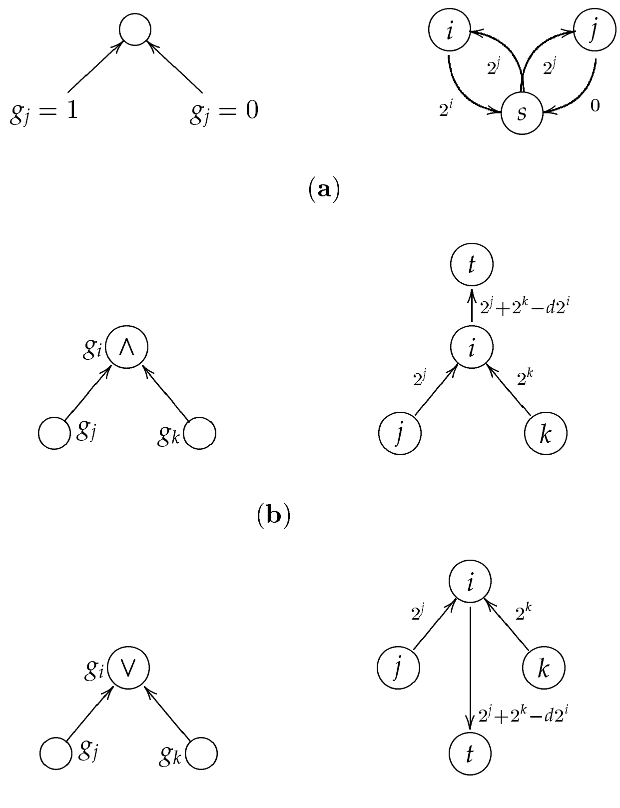
[4]Presná formulácia problému je uvedená v kapitole 2.

1. Pre každý vstupný vrchol g_i obvodu C máme dve hrany (s, i) a (i, s) s kapacitami $c(i, s) = 2^i$
$c(s, i) = \begin{cases} 2^i & \text{ak } g_i = 1; \\ 0 & \text{inak.} \end{cases}$

2. Pre každý AND vrchol obvodu, povedzme $g_i = g_j \wedge g_k$, máme tri hrany $(j, i), (k, i)$ a (i, t). Ich kapacity sú $c(j, i) = 2^j$, $c(k, i) = 2^k$ a $c(i, t) = 2^j + 2^k - d2^i$, kde d je odchádzajúci stupeň (≤ 2) vrcholu g_i.

3. Pre každý OR vrchol obvodu, povedzme $g_i = g_j \vee g_k$, máme tri hrany $(j, i), (k, i)$ a (i, s). Ich kapacity sú $c(j, i) = 2^j$, $c(k, i) = 2^k$ a $c(i, s) = 2^j + 2^k - d2^i$, kde d je odchádzajúci stupeň (≤ 2) vrcholu g_i.

4. Nakoniec máme hranu $(0, t) \in H$ s kapacitou $c(0, t) = 1$



Obrázek 9.3: Hrany zodpovedajúce (a) vstupným vrcholom g_i a g_j, (b) AND vrcholu, (c) OR vrcholu

Konštrukcia grafu G pozostáva vlastne z nahradenia každého vrcholu obvodu nanajvýš tromi hranami. Zrejme takáto konštrukcia sa dá realizovať NC–algoritmom. Zostáva ukázať, že sa skutočne jedná o redukciu problému MCVP2 na problém MAXIMÁLNY TOK.

Lema 9.30 *Maximálny tok v grafe $G = (V, H)$ je nepárny práve ak výstup obvodu C je 1.*

Dôkaz Lemy. Definujeme funkciu $f : H \to \mathbb{N}$. Neskôr o nej ukážeme, že je maximálnym tokom v G. Funkcia f je definovaná takto:

1. Pre každý vstupný vrchol g_i, položíme $f(s,i) = c(s,i)$. Pripomíname, že $c(s,i) = 2^i$ alebo 0, v závislosti od toho, či g_i je 1 alebo 0. Pre hranu (i,s) kladieme $f(i,s) = f(s,i)$ ak g_i nie je vstupom žiadneho vrcholu; inak $f(i,s) = 0$.

2. Pre každú hranu $(i,j) \in H$ takú, že $i,j \notin \{s,t\}$, položíme $f(i,j) = 2^i$ ak hodnota g_i je 1; inak $f(i,j) = 0$.

3. Pre každý AND vrchol $g_i = g_j \wedge g_k$, položíme $f(i,t) = c(i,t)$ ak g_i je rovné 1; inak $f(i,t) = f(j,i) + f(k,i)$.

4. Pre každý OR vrchol $g_i = g_j \vee g_k$, položíme $f(i,s) = f(j,i) + f(k,i) - d2^i$ ak hodnota g_i je 1; inak $f(i,s) = 0$. V obidvoch prípadoch označuje d odchádzajúci stupeň vrchola g_i.

5. Nakoniec položíme $f(0,t) = 1$ ak g_0 je rovné 1; inak $f(0,t) = 0$.

Funkcia f je tokom v grafe G. Ľahko sa overí, že spĺňa kapacitné ohraničenia ako aj podmienky kontinuity.

Funkcia f je vždy maximálnym tokom v G. Predpokladajme opak. Potom musí existovať rezervná cesta Q pre f v G. Cesta Q musí začínať spätnou hranou, lebo kapacita všetkých hrán (s,i) je naplnená. Cesta musí končiť súhlasnou hranou, lebo z vrcholu t nevychádza žiadna hrana. Preto na ceste Q musia existovať tri za sebou nasledujúce vrcholy j,i,k také, že (j,i) je spätná hrana a (j,i) je súhlasná hrana.

Ukážeme, že takáto situácia nemôže nastať. Existujú tri možnosti:

1. g_i je výstupný vrchol. Potom $j = s$, čo implikuje $f(i,s) = 0$; spor.

2. g_i je AND vrchol. Poznamenávame, že (i,j) a (i,k) sú dve hrany, vychádzajúce v vrcholu i a $j \neq t$. Preto g_i je vstupom pre g_j. Potom fakt $f(i,j) > 0$ implikuje $f(i,k) = c(i,k)$; spor.

3. g_i je OR vrchol. Tok f je buď nulový na všetkých hranách vychádzajúcich z OR vrcholu g_i, alebo kapacita všetkých hrán vychádzajúcich z g_i, s možnou výnimkou hrany (i,s), je naplnená. Pretože $k \neq s$ a $f(i,k) < c(i,k)$, tak potom $f(i,j) = 0$, čo je v spore s faktom, že (i,j) je spätná hrana.

To vylučuje existenciu rezervnej cesty pre f, teda f je maximálny tok v G. Pritom parita veľkosti toku závisí len od hodnoty $f(0,t)$, pretože všetky ostatné hrany majú priradené párne hodnoty. Preto výstup obvodu C je 1 práve ak veľkosť toku je nepárna. To dokazuje P–úplnosť problému MAXIMÁLNY TOK. □ □

Ďaľšími príkladmi P–úplných problémov sú

- Problém príslušnosti pre bezkontextové gramatiky
  (pre danú bezkontextovú gramatiku G a slovo w rozhodnúť, či $w \in \mathcal{L}(G)$).
- Problém prázdnosti pre bezkontextové gramatiky
  (pre danú bezkontextovú gramatiku G rozhodnúť, či $\mathcal{L}(G) = \emptyset$).
- Problém konečnosti pre bezkontextové gramatiky
  (pre danú bezkontextovú gramatiku G rozhodnúť, či jazyk $\mathcal{L}(G)$ je konečný).