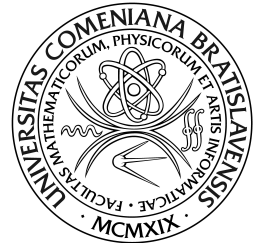




U N I V E R Z I T A K O M E N S K É H O

Fakulta Matematiky, Fyziky a Informatiky

Katedra Informatiky



Pravdepodobnostné algoritmy

Pomocné texty k prednáške 2-INF-134

(verzia 6. apríla 2011)

Bratislava, 2011

Dana Pardubská

Obsah

1	Úvod	5
1.1	Príklad — $AB \stackrel{?}{=} C$	6
1.2	Príklad — Pravdepodobnostný komunikačný protokol	7
1.3	Príklad — Existuje také j , že $x_j = y_j$?	9
2	Klasifikácia náhodou riadených algoritmov	11
2.1	Podľa umiestnenia pravdepodobnosti	11
2.1.1	I. model	11
2.1.2	II. model	13
2.2	Podľa typu a veľkosti chyby	14
2.2.1	Las Vegas	14
2.2.2	Monte Carlo s jednosmernou chybou	17
2.2.3	Monte Carlo s ohraničenou chybou	17
2.2.4	Monte Carlo s neohraničenou chybou	18
2.2.5	Pravdepodobnostné algoritmy pre optimalizačné úlohy	20
3	Pravdepodobnostné zložitostné triedy	25
3.1	Trieda RP	26
3.2	Trieda ZPP	28
3.3	Trieda PP	29
3.4	Trieda BPP	30
3.5	Zdroje náhodných postupností/bitov	33
4	Metódy tvorby pravdepodobnostných algoritmov	41
4.1	Eliminácia protihráča	43
4.1.1	Hašovanie	43
4.1.2	On line algoritmy	47
4.2	Metóda odtlačkov	52
4.2.1	Porovnávanie databáz	52
4.2.2	Freivaldsova metóda - ekvivalencia polynómov	54
4.2.3	Perfect matching	57
4.2.4	Porovnávanie reťazcov	59
4.2.5	Interaktívne dôkazy	61
4.3	Zvyšovanie úspešnosti opakovaním a náhodná vzorka	63
4.3.1	Zlepšovanie úspešnosti opakovaním kritických častí	63
4.3.2	Opakovaná náhodná vzorka a 3-splniteľnosť	69
4.3.3	Náhodná vzorka (random sampling) a generovanie kvadratických nezvyškov	72
4.4	Metóda svedkov	73
4.4.1	Hľadanie svedkov pre test prvočíselnosti	74
4.4.2	Algoritmus Solovay-Strassen pre testovanie prvočíselnosti	76
4.4.3	Generovanie náhodných prvočísel	80

4.5	Optimalizačné úlohy a náhodné zaokrúhľovanie	82
4.5.1	Náhodné zaokrúhľovanie a problém MaxSAT	83
4.5.2	Náhodná vzorka s náhodným zaokrúhľovaním	86
4.6	Semidefinitné programovanie a problém MaxCut	87
5	Ďalšie príklady pravdepodobnostných algoritmov	91
5.1	Paralelné a distribuované výpočty	91
5.1.1	Triedenie na PRAMoch	92
5.1.2	Maximálna nezávislá množina	95
5.1.3	Perfect matching	97
5.1.4	Problém dohody/koordinačný problém	100
5.1.5	Byzantínska dohoda	102
5.1.6	Voľba šéfa	103
5.2	Formálne jazyky a automaty	105
5.2.1	Jednosmerná komunikačná zložitosť	105
5.2.2	Konečné automaty	109
6	Logické obvody	113

Kapitola 1

Úvod

Pojem "náhoda/náhodný" (randomness) je veľmi diskutovaný. Jednou zo základných otázok je otázka, či náhoda naozaj existuje alebo či tento pojem používame na modelovanie javov, ktoré sa vyskytujú nepravidelne/nepredvídateľne (unknown lawfulness).

Demokritos náhoda je neznámo, príroda je určená zákonmi/zákonitosťami.

Epikuros náhoda objektívne existuje, je to prirodzená vlastnosť objektov

do 20teho storočia sa svet vnímal deterministicky, náhoda sa spájala s chaosom a neistotou, strachom; existencia náhodných udalostí sa nepripúšťala. Aj Einstein vinil náhodnosť z nedostatku vedomostí, predpokladal, že každý pravdepodobnostný model fyzikálnej reality by sa dal nahradiť, keby sme mali adekvátne vedomosti.

dnes experimentálna fyzika potvrdila teóriu kvantovej mechaniky, evolučná biológia počíta s náhodnými mutáciami ako hybnou silou evolúcie,...

Alfred Rényi náhoda a poriadok si neprotirečia; viacmenej môžu platiť súčasne. Náhoda kontroluje svet a vďaka tomu panuje vo svete poriadok a zákony, ktoré sa dajú vyjadriť množstvom/meraním náhodných udalostí, ktoré vyplývajú zo zákonov teórie pravdepodobnosti.

Zdá sa, že príroda vždy používa najefektívnejší a najjednoduchší spôsob na dosiahnutie cieľa a že náhoda je prirodzeným konceptom jej stratégie. V praxi máme veľa pravdepodobnostných algoritmov, ktoré sú efektívne a ktorých spoľahlivosť je vysoká, zatiaľčo nepoznáme žiadne deterministické algoritmy, ktorých efektívnosť by bola zrovnateľná. Existujú dokonca prípady, keď adekvátny deterministický algoritmus svojou zložitou presahuje možnosti vesmíru. Dochádza preto k posunu pojmu zvládnuteľných/prakticky riešiteľných (tractable) problémov od deterministického polynomiálneho času k pravdepodobnostnému polynomiálnemu času.

Treba rozlišovať neviem \leftrightarrow nedá sa

- To, že neviem nejaký problém riešiť lepšie znamená, že momentálne nemáme lepší algoritmus. Nehovorí to nič o tom, či lepší v princípe môže existovať.
- Naproti tomu tvrdenie, že sa niečo nedá lepšie riešiť v sebe zahŕňa existenciu dôkazu, že to lepšie nejde.

Formálne sú problémy v NP¹ také, ktoré *nevieme* riešiť deterministickými polynomiálnymi algoritmi. Napriek tomu to v praxi znamená, že k nim pristupujeme

¹NPU označuje triedu NP-úplných problémov. Vieme, že ak $P \neq NP$, čo sa prijíma takmer ako axioma, $NPU \subset NP - P$.

tak, akoby sa riešiteľ lepšie *nedali*.

Randomizácia sa stala štandardným prístupom v návrhu algoritmov. Takéto algoritmy sú zaujímavé hlavne kvôli svojej efektívnosti a jednoduchosti; platíme za to stratou stopercentnej spoľahlivosti - riešenie s malou pravdepodobnosťou nie je korektné, resp. ho nedostaneme.

stochastické \leftrightarrow pravdepodobnostné
pravdepodobnostné sú silnejšie; na každom vstupe sú s veľkou pravdepodobnosťou spoľahlivé

Kľúčovým pojmom pre tvorbu a pochopenie pravdepodobnostných algoritmov je pojem náhody. Väčšinou máme tendenciu povedať, že je to udalosť, ktorú/ktorej výsledok nevieme predpovedať. V tejto súvislosti je dobré si uvedomiť, že hádzanie kockou ani točenie ruletou vlastne náhodnými nie sú - platia tam fyzikálne zákony, takže by sa principiálne dalo vypočítať, čo padne.

1.1 Príklad — $AB \stackrel{?}{=} C$

Uvažujme tri štvorcové matice A, B, C typu $n \times n$. Zaujímá nás, či platí rovnosť

$$AB \stackrel{?}{=} C$$

Deterministický algoritmus založený na Strassenovom násobení matíc je zložitosti $O(n^{\log_2 7})$. Ukážeme, že pravdepodobnostný algoritmus môže byť efektívnejší. Založený je na uvedení si jednoduchého faktu: ak $AB = C$ tak pre ľubovoľný vektor x platí $ABx = Cx$. Alebo aj: ak existuje vektor x , pre ktorý $ABx \neq Cx$, tak $AB \neq C$.

Algoritmus 1 — Test $AB=C$

- 1: náhodne vyber $x \in \{0, 1\}^n$
 - 2: **if** $A(Bx) \neq Cx$ **then** return "určite $AB \neq C$ "
 - 3: **else** return "asi $AB = C$ "
-

Časová zložitosť – vďaka asociativite násobenia (pri existencii dobrého generátora náhodných čísel) sme zložitosť znížili na $O(n^2)$.

Chyba – chyby sa dopustíme v tom prípade, ak $AB \neq C$ a pritom pre náhodne zvolený vektor x platí $ABx = Cx$. Odhadnime pravdepodobnosť toho, že sa tak stane.

Lema 1.1 *Nech $AB \neq C$, $x \in \{0, 1\}^n$. Potom $Pr[ABx = Cx] \leq 1/2$*

Dôkaz: Ak $AB \neq C$, potom $AB - C \neq 0$. Existujú teda indexy i, j tak, že $(AB - C)[i, j] \neq 0$. Nech (a_1, \dots, a_n) označuje i -ty riadok matice $AB - C$. Potom

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

Využili sme, že $a_j \neq 0$ a $Pr[x_i = 0] = Pr[x_i = 1] = 1/2$

Ak algoritmus skončí s odpoveďou "asi", môžeme ho zopakovať. Pri k -násobnom zopakovaní sa časová zložitosť zvýši o multiplikatívnych k . Pritom k - násobné

zopakovanie algoritmu — k -násobné získanie odpovede "asi" — znamená chybu nanajvyš $1/2^k$. □

Uvedený algoritmus je rýchly, jedným smerom — $AB \neq C$ — hovorí korektne, chyby sa môže dopustiť len pri odpovedi "asi $AB = C$ ". Takýto typ algoritmu sa volá Monte Carlo.

1.2 Príklad — Pravdepodobnostný komunikačný protokol

Majme dva vzdialené počítače RI a RII, každý z nich udržiava "databázu údajov", čo je pre nás binárny reťazec (resp. v prípade potreby binárne číslo) $X = x_1x_2 \dots x_n$, resp. $Y = y_1y_2 \dots y_n$. Chceme vedieť, či sú tieto "databázy" totožné, pričom sa snažíme o minimalizáciu komunikácie medzi počítačmi.

Ak chceme problém riešiť deterministicky, lepšie ako poslanie celej informácie jedného počítača tomu druhému v princípe nenájde. Dá sa to aj (napr. pomocou Kolmogorovskej alebo komunikačnej zložitosti) dokázať.

Jednoduchý pravdepodobnostný algoritmus využíva zvyšky po delení prvočíslom — ak sú dva čísla X, Y rovnaké, potom sú rovnaké aj ich zvyšky po delení ľubovoľným (prvo)číslom. Pre náhodne zvolené prvočíslo p si počítače RI a RII porovnajú zvyšky $X \bmod p$ a $Y \bmod p$. Ak sú zvyšky rôzne, čísla X, Y určite nie sú rovnaké. Ak sú však X, Y rôzne, môže sa stať, že sa napriek tomu zvyšky rovnajú. Ukážeme, že pravdepodobnosť takejto situácie je malá, nanajvyš $1/2$.

Algoritmus 2 — Test $X=Y$

RI:

- 1: náhodne vyber prvočíslo p z množiny $\{2, \dots, n^2\}$;
 - ▷ keďže sa v tomto intervale nachádza $\sim n^2 / \ln n^2$ prvočísel, stačí na reprezentáciu náhodného prvočísla $\lceil \log_2 n^2 \rceil \leq 2 \cdot \lceil \log_2 n \rceil$ bitov
- 2: $s \leftarrow X \bmod p$
- 3: pošli s druhému počítaču RII
 - ▷ dĺžka správy je nanajvyš $4 \cdot \lceil \log_2 n \rceil$ bitov

RII:

- 1: $s' \leftarrow Y \bmod p$
 - 2: **if** $s = s'$ **then** databázy prehlásime za rovnaké
 - 3: **else** databázy prehlásime za rôzne
-

Pri $n = 10^{16}$ je počet prekomunikovaných bitov nanajvyš $4 \cdot 16 \lceil \log_2 10 \rceil = 256$. Túto krátku správu zrejme dokážeme bezpečne prekomunikovať.

A teraz sa zamyslime nad korektnosťou algoritmu. Je zrejmé, že negatívna odpoveď je vždy pravdivá. Ostáva spočítať, s akou pravdepodobnosťou sme sa dopustili chyby v prípade pozitívnej odpovede.

$x \neq y$, databázy sme prehlásili za rovnaké Nesprávnu odpoveď dostaneme v prípade, ak pre náhodne zvolené prvočíslo p platí

$$z = X \bmod p = Y \bmod p$$

$$X = x \cdot p + z, Y = y \cdot p + z \quad \implies \Delta = X - Y = (x - y) \cdot p$$

Takže "zlé" je také prvočíslo p , ktoré delí číslo $\Delta = X - Y < 2^n$. Počet potenciálne "zlých" prvočísel odhadneme pomocou faktorizácie čísla Δ

$$\Delta = p_1^{i_1} \dots p_k^{i_k}$$

kde $p_1 < p_2 < \dots < p_k$ sú tie prvočísla, pre ktoré $i_j \neq 0$. Platí

$$2^n > \Delta = p_1^{i_1} \dots p_k^{i_k} \geq p_1 p_2 \dots p_k > 1 \cdot 2 \cdot 3 \cdot \dots \cdot k = k!$$

a preto pre počet k zlých prvočísel máme $k \leq n-1$. Keďže pravdepodobnosť výberu konkrétneho prvočísla je pre všetky prvočísla v sledovanom intervale rovnaká, je pravdepodobnosť výberu "zlého" prvočísla, ktoré delí Δ , nanajvyšš

$$\frac{n-1}{\text{Prim}(n^2)} \tag{1.1}$$

kde $\text{Prim}(n^2)$ označuje počet prvočísel nanajvyšš veľkosti n^2 . Predstavu o hodnote $\text{Prim}(m)$ dáva Lema 1.2, ktorú uvádzame bez dôkazu.

$\text{Prim}(m)$

Lema 1.2 *Nech $\text{Prim}(m)$ označuje počet prvočísel v množine $\{2, \dots, m\}$. Potom*

$$\lim_{m \rightarrow \infty} \frac{\text{Prim}(m)}{m / \ln m} = 1 \qquad \text{Prim}(m) > \frac{m}{\ln m}, \quad m > 67$$

Spojením (1.1) a Lemy 1.2 dostávame, že pravdepodobnosť chybnnej odpovede je

$$\frac{n-1}{\text{Prim}(n^2)} \leq \frac{n-1}{n^2 / \ln n^2} \leq \frac{2 \ln n}{n}$$

čo pre $n = 10^{16}$ dáva hodnotu nanajvyšš $0,36892 \cdot 10^{-14}$.

Pravdepodobnosť chybnnej odpovede môžeme znížiť jednoduchým zásahom do algoritmu - algoritmus niekoľkokrát nezávisle zopakujeme. Namiesto jedného prvočísla p zvolíme *nezávisle* niekoľko prvočísel p_1, \dots, p_k . Potom nesprávnu odpoveď získame len vtedy, ak pre každé i bude p_i deliť rozdiel $X - Y$. O chybe potom zrejme platí

$$\left(\frac{n-1}{\text{Prim}(n^2)} \right)^k \leq \left(\frac{\ln n^2}{n} \right)^k = \frac{2^k \cdot (\ln n)^k}{n^k}$$

Pre $n = 10^{16}$ a $k = 10$ máme pravdepodobnosť chyby nanajvyšš $0,4717 \cdot 10^{-141}$. Pritom dĺžka prekomunikovanej postupnosti sa zmenila o multiplikatívnu konštantu k (v našom prípade desaťnásobne).

Cvičenie 1.1 *Pravdepodobnosť chyby nášho algoritmu by sme mohli znížiť aj iným prístupom - zmeníme protokol tak, že nebudeme náhodne vyberať prvočíslo z intervalu $[2, n^2]$, ale z intervalu $[2, n^r]$. Odvodte dĺžku prekomunikovaného reťazca a pravdepodobnosť chyby takto modifikovaného algoritmu pre $r \geq 2$.*

Cvičenie 1.2 *Nech $\delta > 1$. Navrhnite pravdepodobnostný protokol na porovnanie dvoch databáz veľkosti n , ktorý pracuje s pravdepodobnosťou chyby nanajvyšš $1/\delta$.*

Cvičenie 1.3 *Porovnajte prístupy z cvičenie 1.1 a 1.2. Ktorý je efektívnejší z hľadiska dĺžky prekomunikovanej postupnosti? Je lepšie zvoliť viacero krátkych prvočísel alebo jedno dlhé?*

1.3 Príklad — Existuje také j, že $x_j = y_j$?

Tentokrát si naše vzdialené počítače udržiavajú postupnosť, povedzme 10, reťazcov

$$\begin{array}{ll} \text{RI} & x_1, \dots, x_{10} \quad x_i \in \{0, 1\}^n \\ \text{RII} & y_1, \dots, y_{10} \quad y_i \in \{0, 1\}^n \end{array}$$

Zaujímá nás, či existuje taký index j , aby $x_j = y_j$. Opäť sa dá dokázať, že deterministický protokol vyžaduje prekomunikovanie informácie veľkosti $10n$. Pritom pravdepodobnostnému algoritmu, opäť využívajúcemu zvyšky po delení prvočíslom, stačí prekomunikovať menej.

Nášmu algoritmu dovolíme, aby v situácii, keď si svojou odpoveďou nie je istý, povedal "neviem". Vyžadujeme však, aby to nerobil príliš často. Takému typu algoritmov hovoríme Las Vegas.

Algoritmus 3 — Existuje j , $x_j = y_j$?

RI:

- 1: náhodne zvol 10 prvočísel p_1, \dots, p_{10}
- 2: $s_i \leftarrow x_i \bmod p_i$
- 3: pošli $p_1, \dots, p_{10}, s_1, \dots, s_{10}$ počítaču RII

RII:

- 1: $s'_i \leftarrow y_i \bmod p_i$
- 2: **if** $s_i \neq s'_i$ pre všetky i **then** return "nie"
- 3: **else** nech j je minimálne také, že $s_j = s'_j$
- 4: pošli počítaču RI j, y_j

RI:

- 1: **if** $x_j = y_j$ **then** odpoveď je "áno"
 - 2: **else** odpoveď je "neviem"
-

Lahko vidno, že počítače prekomunikovali $20(2\lceil \log n \rceil) + \log 10 + n$ bitov.

Chyba — analýzu algoritmu rozdelíme na dva prípady podľa toho, či hľadaný index j existuje alebo nie.

Začnime prípadom, keď korektná odpoveď algoritmu má byť "nie". Už vieme, že napriek tomu, že čísla X, Y dĺžky n sú rôzne, s pravdepodobnosťou najviac $\frac{2 \ln n}{n}$ sa môže stať, že pre náhodne zvolené prvočíslo $p \leq n^2$ platí $X \bmod p = Y \bmod p$. My volíme náhodne 10 prvočísel, preto s pravdepodobnosťou $(1 - \frac{2 \ln n}{n})^{10}$ sa to nestane pre žiadne z nich; vtedy RII bez ďalšej komunikácie s RI korektne odpovie "nie". Pre pravdepodobnosť Pr korektnej odpovede (v prípade $x_i \neq y_i \forall i$) preto platí

$$\boxed{\forall i \ x_i \neq y_i}$$

$$\begin{aligned} Pr &\geq \left(1 - \frac{2 \ln n}{n}\right)^{10} = \sum_{i=0}^{10} -1^i \binom{10}{i} \left(\frac{2 \ln n}{n}\right)^i \\ &= 1 + \sum_{i=1}^5 \binom{10}{2i} \left(\frac{2 \ln n}{n}\right)^{2i} - \sum_{i=0}^4 \binom{10}{2i+1} \left(\frac{2 \ln n}{n}\right)^{2i+1} \\ &= 1 - \binom{10}{1} \frac{2 \ln n}{n} + \binom{10}{10} \left(\frac{2 \ln n}{n}\right)^{10} + \\ &\quad + \underbrace{\sum_{i=1}^4 \left[\binom{10}{2i} \left(\frac{2 \ln n}{n}\right)^{2i} - \binom{10}{2i+1} \left(\frac{2 \ln n}{n}\right)^{2i+1} \right]}_{\geq 0} \\ &\geq 1 - \frac{20 \ln n}{n} \geq 1/2 \end{aligned}$$

Nech j je najmenšie také, že $x_j = y_j$. Označme E_j udalosť, keď j je najmenšie také, že $x_j \bmod p_j = y_j \bmod p_j$; inými slovami, RI pošle RI index j a reťazec y_j . $\exists j \mathbf{x}_j = \mathbf{y}_j$

- ak $j = 1$, RI úspešne overí, že $x_1 = y_1$ a korektne odpovie
- ak $j > 1$, je nenulová pravdepodobnosť, že $x_j \neq y_j$. Pravdepodobnosť korektnej odpovede je rovná pravdepodobnosti udalosti E_j , čo je aspoň

$$\left(1 - \frac{2 \ln n}{n}\right)^{j-1} \geq 1 - \frac{2(j-1) \ln n}{n} \geq 1 - \frac{18 \ln n}{n} \geq 1/2, \text{ pre } n \geq 189$$

Nezávisle od toho, či hľadaný index existuje alebo nie, dá algoritmus s pravdepodobnosťou aspoň $1/2$ korektnú odpoveď, v ostatných prípadoch povie "neviem".

Kapitola 2

Klasifikácia náhodou riadených algoritmov

2.1 Podľa umiestnenia pravdepodobnosti

Jeden z možných pohľadov na klasifikáciu pravdepodobnostných algoritmov je ten, keď si všimáme umiestnenie pravdepodobnosti v nich.

- I. modelom pravdepodobnostného algoritmu je pravdepodobnostné rozdelenie nad množinou deterministických stratégií
- II. pravdepodobnostný algoritmus modelujeme nedeterministickým algoritmom s pravdepodobnostným rozdelením nad nedeterministickými voľbami

2.1.1 I. model

Pravdepodobnostný algoritmus vnímame ako pravdepodobnostné rozdelenie $Prob$ nad konečnou množinou deterministických stratégií $\{A_1, \dots, A_m\}$.

- Pre vstup w náhodne zvolíme i a realizujeme výpočet $C_i = A_i(w)$, ktorého časová zložitosť je $Time(C_i)$; náhodná voľba i odpovedá pravdepodobnostnému rozdeleniu $Prob$.
- Pravdepodobnostný priestor, v ktorom sa hýbeme, je $(S_{A,w}, Prob)$, kde $S_{A,w} = \{C_1, \dots, C_m\}$.

Čas je vlastne náhodná premenná.

$$\mathbf{Z}: S_{A,w} \rightarrow \mathbb{N}$$
$$Z(C_i) = Time(C_i)$$

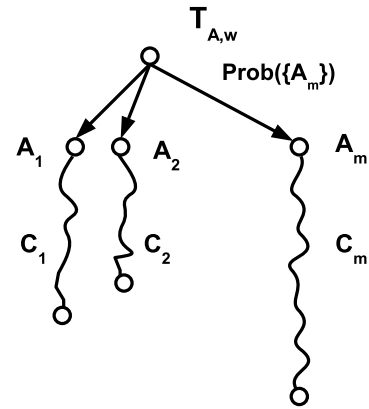
$$\mathbf{ExpTime}_A(\mathbf{w}) = \sum_{i=1}^m Prob[C_i] \cdot Z(C_i)$$
$$= \sum_{i=1}^m Prob[C_i] \cdot Time(C_i)$$

$$\mathbf{ExpTime}_A(\mathbf{n}) = \max_{|w|=n} \{ExpTime_A(w)\}$$

$$\mathbf{Time}_A(\mathbf{n}) = \max_{i, |w|=n} \{Time(A_i(w))\}$$

$$\mathbf{X}(C_i) = \begin{cases} 1, & \text{ak výpočet } C_i \text{ dáva korektnú odpoveď;} \\ 0, & \text{inak} \end{cases}$$

Oproti deterministickým algoritmom pribúda pojem očakávaná zložitosť/očakávaný prípad, ktorý v sebe zahŕňa všetky možné výpočty na jednotlivých slovách.



Pravdepodobnosť korektnosti

$$\begin{aligned}
E[X] &= \sum_{i=1}^m X(C_i) \cdot Prob[C_i] \\
&= \sum_{X(C_i)=1} 1 \cdot Prob[C_i] + \sum_{X(C_i)=0} 0 \cdot Prob[C_i] \\
&= Prob[X = 1] = Prob[A \text{ počíta korektný výstup}]
\end{aligned}$$

Pravdepodobnosť chyby

$$Error_A(w) = 1 - E[X]$$

$$Error_A(n) = \max_{|w|=n} \{Error_A(w)\}$$

Príklad 2.1 Porovnanie "databáz"

- $S_{R,(x,y)} = \{C_p \mid p \in Prim(n^2)\}$
- rovnomerné rozdelenie
- $X = Y \Rightarrow$ žiadna chyba
- $X \neq Y \Rightarrow X(C_p) = \begin{cases} 1, & p \text{ je "dobré"} \\ 0, & p \text{ je "zlé"} \end{cases}$
- $Prob[C_p] = \frac{1}{Prim(n^2)}$
- $E[X] = \sum_{p \in Prim(n^2)} X(C_p) \cdot Prob[C_p]$
 $= \sum_{p \in Prim(n^2)} X(C_p) \cdot \frac{1}{Prim(n^2)} = \frac{1}{Prim(n^2)} \sum_{p \text{ je dobre}} X(C_p) \geq$
 $\geq \frac{1}{Prim(n^2)} (Prim(n^2) - (n-1)) = 1 - \frac{n-1}{Prim(n^2)}$
- $Error_R((x,y)) = 1 - E[X] \leq \frac{n-1}{Prim(n^2)} \leq \frac{2 \ln n}{n}$

Príklad 2.2 (MAX-SAT) Pre vstupnú formulu $\Phi(x_1, \dots, x_n)$ v KNF chceme vypočítať také priradenie vstupných hodnôt $\alpha \in \{0, 1\}^n$, ktoré spĺňa maximálny počet klauzúl.

Ukážeme, že náhodný vektor α spĺňa s veľkou pravdepodobnosťou "dost" klauzúl. Preto nasledujúci algoritmus RSAM má zmysel.

Algoritmus 4 RSAM

vstup: $\Phi = F_1 \wedge F_2 \wedge \dots \wedge F_m$, $F_i = (l_{i,1} \vee l_{i,2} \vee \dots \vee l_{i,k})$

1: náhodne $\alpha = (\alpha_1, \dots, \alpha_n) \in \{0, 1\}^n$

$\triangleright Pr(\alpha_j = 1) = Pr(\alpha_j = 0) = 1/2$

2: return(α)

Zaujímá nás kvalita uvedeného algoritmu. Zoberieme preto náhodnú premennú z , ktorá počíta počet splnených klauzúl, a vypočítame jej strednú hodnotu $E[Z]$.

$$Z_i(\alpha) = \begin{cases} 1, & F_i(\alpha) = 1 \\ 0, & F_i(\alpha) = 0 \end{cases}$$

$$Z = \sum_{i=1}^m Z_i$$

$$E[Z] = E \left[\sum_{i=1}^m Z_i \right] = \sum_{i=1}^m E[Z_i]$$

K výpočtu $E[Z]$ odhadneme $E[Z_i]$:

- $F_i(\alpha) = 0 \iff \forall j \ l_{i,j} = 0$
 $Pr[F_i(\alpha) = 0] = \left(\frac{1}{2}\right)^k = \frac{1}{2^k}$
- $E[Z_i] = 1 - \frac{1}{2^k}$; keďže formula má aspoň jeden literál, $E[Z_i] \geq 1/2$
- $E[Z] = \sum_{i=1}^m E[Z_i] \geq \sum_{i=1}^m 1/2 = \frac{m}{2}$

Keďže vieme, že očakávaný počet klauzúl, ktoré náhodný vektor spĺňa, je aspoň $m/2$, môžeme to využiť na jednoduché hľadanie vektora, ktorý spĺňa aspoň $m/2$ klauzúl.

Algoritmus 5 modif-RSAM

```

1: náhodne  $\alpha = (\alpha_1, \dots, \alpha_n) \in \{0, 1\}^n$ 
2:  $r \leftarrow |\{i \mid F_i(\alpha) = 1\}|$ 
3: if  $r \geq m/2$  then return( $\alpha$ )
4: else goto 1
  
```

2.1.2 II. model

Pri tomto type priradíme pravdepodobnosť jednotlivým nedeterministickým voľbám. Príkladom je pravdepodobnostný Quicksort RQS

Algoritmus 6 RQS(A)

```

vstup:  $A = \{a_1, \dots, a_n \mid a_i \in (S, <)\}$ 
1: if  $A = \{a\}$  then return  $a$ 
2: else  $b \leftarrow \text{random}(A)$ 
3:  $A_< = \{a \in A \mid a < b\}$ 
4:  $A_> = \{a \in A \mid a > b\}$ 
5: return  $RQS(A_<), b, RQS(A_>)$ 
  
```

čas je počet porovnaní

- vždy extrém, potom $O(n^2)$
- vždy medián, potom $O(n \log n)$
- riešenie $T(n) \leq T(n/8) + T(7n/8) + n - 1$ je tiež $O(n \log n)$; takéto rozdelenie je dosť pravdepodobné

Nech výstupom je $s_1 < s_2 < \dots < s_n$. Pre $i < j$ označme

$$X_{i,j}(C) = \begin{cases} 1, & s_i \text{ sa v priebehu } C \text{ porovnávalo s } s_j \\ 0, & s_i, s_j \text{ sa v priebehu } C \text{ neporovnávajú} \end{cases}$$

$$T(C) = \sum_{i=1}^n \sum_{j>i} X_{i,j}(C)$$

$$E[T] = E \left[\sum_{i=1}^n \sum_{j>i} X_{i,j} \right] = \sum_{i=1}^n \sum_{j>i} E[X_{i,j}]$$

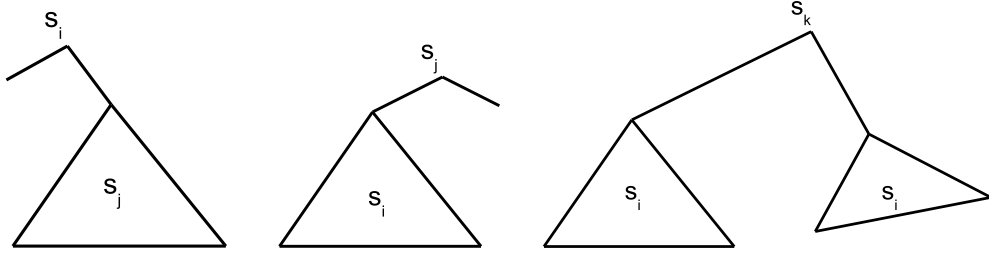
Ohraničme $E[X_{i,j}]$

$p_{i,j}$ —pravdepodobnosť, že sa s_i, s_j porovnávajú

$$E[X_{i,j}] = p_{i,j}$$

$$\underbrace{s_1 \ \dots \ s_{i-1}}_L \ s_i \ \underbrace{\dots \ s_j}_M \ \underbrace{s_{j+1} \ \dots \ s_n}_R$$

s_i sa porovnáva s s_j v takých výpočtoch, keď jeden z x_i, x_j je ako pivot zvolený skôr, ako ktorýkoľvek z prvkov v M



$$p_{i,j} = \frac{|\{s_i, s_j\}|}{|M \cup \{s_i, s_j\}|} = \frac{2}{j-i-1+2} = \frac{2}{j-i+1}$$

$$\begin{aligned} \mathbf{E}[\mathbf{T}] &= \sum_{i=1}^n \sum_{j>i} p_{i,j} = \sum_{i=1}^n \sum_{j>i} \frac{2}{j-i+1} \\ &= \sum_{i=1}^n \sum_{k=2}^{n-i+1} \frac{2}{k} \\ &= 2 \sum_{i=1}^n \sum_{k=1}^{n-i+1} \frac{1}{k} \\ &\leq 2 \sum_{i=1}^n H(n) = \mathbf{O}(n \ln n) \end{aligned}$$

$$H(n) = \underbrace{\frac{1}{1}}_{<1} + \underbrace{\frac{1}{2} + \frac{1}{3}}_{<1} + \underbrace{\frac{1}{4} + \frac{1}{5} + \frac{1}{6} + \frac{1}{7}}_{<1} + \dots + \frac{1}{n} \quad \ln n \text{ skupín so súčtom } < 1$$

2.2 Podľa typu a veľkosti chyby

2.2.1 Las Vegas

Označme $A(x)$ odpoveď algoritmu A a $F(x)$ korektnú odpoveď na vstupe w .

D1

Definícia 2.1 Pravdepodobnostný algoritmus A je typu Las Vegas (LV) keď

$$Pr[A(x) = F(x)] = 1$$

Pri tejto definícii nás zaujíma očakávaná zložitosť ExpTime.

D2

Definícia 2.2 Pravdepodobnostný algoritmus A je typu Las Vegas (LV) keď $\forall x$

- $Pr[A(x) = F(x)] \geq 1/2$
- $Pr[A(x) = \text{"?"}] = 1 - Pr[A(x) = F(x)] \leq 1/2$

Konštanta $1/2$ nie je podstatná, vyhovuje ľubovoľné $0 < \varepsilon < 1$. Zamyslime sa, aký je vzťah medzi týmito dvomi definíciami. Ukážeme, že sú ekvivalentné.

Lema 2.1 Ku každému LV algoritmu $A_?$ v zmysle definície 2.2 existuje ekvivalentný algoritmus A , ktorý je LV v zmysle definície 2.1. Navyše, $ExpTime_A(n) = O(ExpTime_{A_?}(n))$

Dôkaz: Samotná konštrukcia ekvivalentného A je priamočiara—budeme opakovať výpočet $A_?$ dovtedy, kým nedostaneme korektnú odpoveď.

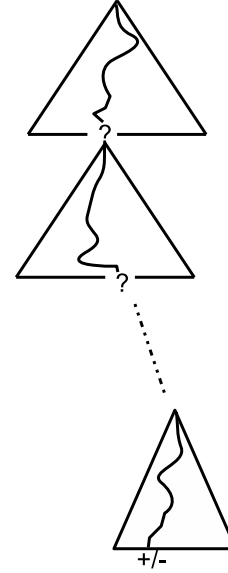
pravdepodobnosť korektnej odpovede

Čo vieme povedať o pravdepodobnosti získania korektnej odpovede A , resp. dosiahnutia odpovede "?" algoritmu $A_?$? Nech p je pravdepodobnosť korektnej odpovede algoritmu $A_?$, $p \geq 1/2$. Potom pravdepodobnosť toho, že po k opakovaní behu algoritmu $A_?$ ešte stále nemáme korektnú odpoveď, je nanajvýš $1/2^k$. Preto $Prob[A_1(x) = F(x)] = 1$

čas

Je zrejmé, že A môže realizovať nekonečné výpočty. Ukážeme, že v očakávanom prípade je situácia oveľa lepšia. Uspokojíme sa s horným odhadom na očakávaný prípad, preto budeme bez ujmy na všeobecnosti predpokladať, že každý výpočet algoritmu $A_?$, ktorý končí odpoveďou "?", dosahuje zložitosť najhoršieho prípadu.

Všimnime si výpočty, ktoré *skončia* v i -tom kole.



- Nech $Set_i = \{C \in S_{A,w} \mid (i-1)Time_{A_?} < Time_A(C) \leq iTime_{A_?}(w)\}$
Zrejme $S_{A,w} = \cup_{i=0}^{\infty} Set_i$, $\forall i \neq j \ Set_i \cap Set_j = \emptyset$
- Aby výpočet C mohol skončiť v i -tom kole, musí $(i-1)$ kôl skončiť odpoveďou "?". Pravdepodobnosť odpovede "?" je nanajvýš $1/2$, preto pravdepodobnosť, že výpočet neskončí po $i-1$ kolách je nanajvýš $(\frac{1}{2})^{i-1}$. Z toho dostávame

$$\sum_{C \in Set_i} Prob[C] \leq \frac{1}{2^{i-1}}$$

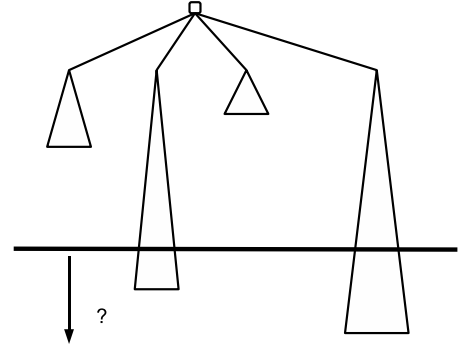
$$\begin{aligned} ExpTime_A(w) &= \sum_{C \in S_{A,w}} Time_A(C) \cdot Prob[C] \\ &= \sum_{i=1}^{\infty} \sum_{C \in Set_i} Time_A(C) \cdot Prob[C] \\ &\leq \sum_{i=1}^{\infty} \sum_{C \in Set_i} iTime_{A_?}(w) \cdot Prob[C] \\ &= \sum_{i=1}^{\infty} iTime_{A_?}(w) \cdot \sum_{C \in Set_i} Prob[C] \\ &\leq \sum_{i=1}^{\infty} iTime_{A_?}(w) \cdot \frac{1}{2^{i-1}} \\ &= Time_{A_?}(w) \cdot \sum_{i=1}^{\infty} \frac{i}{2^{i-1}} \\ &= Time_{A_?}(w) \cdot 2 \cdot \sum_{i=1}^{\infty} \frac{i}{2^i} = 4 \cdot Time_{A_?}(w) \end{aligned}$$

□

Lema 2.2 *Ku každému LV algoritmu A v zmysle definície 2.1 existuje ekvivalentný algoritmus $A_?$, ktorý je LV v zmysle definície 2.2.*

Dôkaz:

Ak do korektného algoritmu ideme vniesť nejednoznačnosť/odpoveď neviem, tak je prirodzené vyžadovať, aby ten nový algoritmus bol aspoň rýchly. Takto budeme konštruovať požadovaný algoritmus $A_?$. Zvolíme rozumnú hranicu a všetky výpočty, ktoré dovedy neskončili, "ustrihne" s odpoveďou "?".



Pri voľbe hranice pre odpoveď "?" treba mať na pamäti, že podľa definície sa pod touto hranicou môže ocitnúť najviac polovica všetkých výpočtov. Keďže najviac polovica čísel má hodnotu väčšiu ako dvojnásobok priemeru, bude rozumným ohraničením

$$T = 2ExpTime_A(w)$$

Kvôli sporu predpokladajme, že $Prob[A_?(w) = "?"] > 1/2$. Označme

- $S_{A,w}(?)$ množinu výpočtov s odpoveďou "?"
 $S_{A,w}(?) = \{C \in S_{A,w} \mid Time(C) > 2ExpTime_A(w)\}$
- $S_{A,w}(F(w))$ množinu výpočtov s korektnou odpoveďou $F(w)$
 $S_{A,w}(F(w)) = S_{A,w} - S_{A,w}(?)$

Platí:

- $Prob[A_?(w) = "?"] = \sum_{C \in S_{A,w}(?)} Prob[C] > \frac{1}{2}$
- Všetky výpočty z $S_{A,w}(?)$ sú dlhé: $Time(C) \geq 2ExpTime_A(w) + 1$.

Preto

$$\begin{aligned} \mathbf{ExpTime_A(w)} &= \sum_{C \in S_{A,w}} \overbrace{Time_A(C) Prob[C]}^* \\ &= \sum_{C \in S_{A,w}(?)} * + \sum_{C \in S_{A,w}(F(w))} * \\ &\geq \sum_{C \in S_{A,w}(?)} (2ExpTime_A(w) + 1) \cdot Prob[C] + 0 \\ &= (2ExpTime_A(w) + 1) \underbrace{\sum_{C \in S_{A,w}(?)} Prob[C]}_{>1/2} \\ &> \frac{(2ExpTime_A(w) + 1)}{2} = \mathbf{ExpTime_A(w) + 1/2} \end{aligned}$$

□

trieda LV^*

Trieda LV algoritmov, pre ktoré $Prob[A(x) = F(x)] = 1$, sa niekedy označuje Las Vegas*, resp. LV^* ; hviezdica zrejme naznačuje možnosť nekonečných výpočtov.

2.2.2 Monte Carlo s jednosmernou chybou

O tejto triede algoritmov ¹ hovoríme v súvislosti s rozhodovacími problémami². Neformálne ide o také algoritmy, ktoré majú povolené sa mýliť/klamať len jedným smerom.

Definícia 2.3 *Pravdepodobnostný algoritmus A pre rozhodovací problém (Σ, L) je 1MC Monte Carlo s jednosmernou chybou/jednosmerný Monte Carlo (1MC, one-sided MC), ak*

- (a) $\forall x \in L \quad Prob[A(x) = 1] \geq 1/2$
- (b) $\forall x \notin L \quad Prob[A(x) = 0] = 1$

Keď podmienku (a) nahradíme podmienkou

- (a*) $\forall x \in L \quad Prob[A(x) = 1] \rightarrow 1$ s rastúcou dĺžkou $|x|$

dostaneme tzv. 1MC* algoritmy

1MC*

Príkladom 1MC pravdepodobnostného algoritmu je algoritmus 2 na porovnanie databáz, resp. algoritmus 1 na pravdepodobnostné porovnanie matic $?AB = C?$.

Ako sme videli, pravdepodobnosť chyby klesá exponenciálne s počtom *nezávislých znižovanie chyby* opakovaní. Ak je teda 1MC algoritmus efektívny, ľahko z neho získame algoritmus rádoovo rovnakej zložitosti s exponenciálne menšou chybou.

- nech $\alpha_1, \dots, \alpha_k \in \{0, 1\}^k$ je k výsledkov nezávislých behov 1MC algoritmu A
- ak $\exists k : \alpha_k = 1$, môžeme so 100% istotou akceptovať
- ak $\alpha_1 = \dots = \alpha_k = 0$, tak pre $x \in L$ je $Prob[A(x) = 0] \leq 1/2$ a teda $(Prob[A(x) = 0])^k \leq 2^{-k}$.

2.2.3 Monte Carlo s ohraničenou chybou

V prípade takého problému, ktorý nie je rozhodovací, ale je to problém počítania funkcie $F(x)$, pod korektnou odpoveďou prirodzene rozumieme, že algoritmus vypočítal presne hodnotu $F(x)$; nekorektnou odpoveďou je nepresný výsledok. Pri Monte Carlo algoritmoch s ohraničenou chybou³ pripustíme, aby algoritmus dával nesprávnu odpoveď, budeme ale vyžadovať, aby bol "dostatočný" rozdiel medzi odpoveďou korektnou a nekorektnou. Formálne:

Definícia 2.4 *Pravdepodobnostný algoritmus A je Monte Carlo s ohraničenou chybou (2MC, bounded-error MC), ak*

$$\exists 0 < \varepsilon \leq 1/2 \quad \forall x \quad Prob[A(x) = F(x)] \geq 1/2 + \varepsilon$$

V prípade 1MC algoritmov sme pravdepodobnosť chyby jednoducho znížili nezávislým opakovaním algoritmu. Ako nám pomôže nezávislé opakované spúšťanie 2MC algoritmu? *znižovanie chyby*

Uvažujme t nezávislých opakovaní 2MC algoritmu A a odpovedzme len vtedy, ak sa na odpovedi zhodne aspoň $t/2$ behov (algoritmus 7). Zaujímá nás, aká je pravdepodobnosť toho, že *nedostaneme* korektnú odpoveď.

Keďže A je 2MC, existuje $0 < \varepsilon \leq 1/2$ $Prob[A(x) = F(x)] \geq 1/2 + \varepsilon$. Nech x je vstup. Označme

- $p = p(x) = Prob[A(x) = F(x)] = 1/2 + \varepsilon_x, \varepsilon_x \geq \varepsilon$

¹One-sided Monte Carlo

²Rozhodovací problém je dvojica (Σ, L) , pričom sa pre $x \in \Sigma^*$ pýtame, či $?x \in L?$

³Bounded-Error Monte Carlo

Algoritmus 7 A_t

-
- 1: nech $\alpha_1, \dots, \alpha_t$ je výsledok t nezávislých behov 2MC algoritmu A
 - 2: ak existuje taký výsledok α , ktorý sa vyskytol aspoň $t/2$ krát, výsledkom je " α ", inak je výsledkom "?"
-

- $pr_i(x)$ pravdepodobnosť toho, že spomedzi k opakovaní $A(x)$ je presne i odpovedí korektných; je presne $\binom{k}{i}$ možností takýchto výpočtov

Pre určenie chyby algoritmu 7 využijeme odhad na $pr_i(x)$; algoritmus A_k neodpovie korektne, ak menej ako $k/2$ jednotlivých behov odpovie korektne.

$$\begin{aligned}
pr_i(x) &= \binom{k}{i} p^i (1-p)^{k-i} = \binom{k}{i} [p(1-p)]^i (1-p)^{k-2i} \\
&= \binom{k}{i} \left[\left(\frac{1}{2} + \varepsilon_x \right) \left(\frac{1}{2} - \varepsilon_x \right) \right]^i \left(\frac{1}{2} - \varepsilon_x \right)^{k-2i} \\
&= \binom{k}{i} \left(\frac{1}{4} - \varepsilon_x^2 \right)^i \left[\left(\frac{1}{2} - \varepsilon_x \right)^2 \right]^{k/2-i} \\
&< \binom{k}{i} \left(\frac{1}{4} - \varepsilon_x^2 \right)^i \left[\left(\frac{1}{2} + \varepsilon_x \right) \left(\frac{1}{2} - \varepsilon_x \right) \right]^{k/2-i} \\
&= \binom{k}{i} \left(\frac{1}{4} - \varepsilon_x^2 \right)^i \left(\frac{1}{4} - \varepsilon_x^2 \right)^{k/2-i} = \binom{k}{i} \left(\frac{1}{4} - \varepsilon_x^2 \right)^{k/2} \\
&\leq \binom{k}{i} \left(\frac{1}{4} - \varepsilon_x^2 \right)^{k/2}
\end{aligned}$$

A_k odpovie korektne, ak aspoň $\lceil k/2 \rceil$ jednotlivých behov odpovie korektne.

$$\begin{aligned}
\text{Prob}[A_k(x) = F(x)] &= \sum_{i=\lceil k/2 \rceil}^k pr_i(x) = 1 - \sum_{i=0}^{\lfloor k/2 \rfloor} pr_i(x) \\
&> 1 - \sum_{i=0}^{\lfloor k/2 \rfloor} \binom{k}{i} \left(\frac{1}{4} - \varepsilon_x^2 \right)^{k/2} \\
&= 1 - \left(\frac{1}{4} - \varepsilon_x^2 \right)^{k/2} \sum_{i=0}^{\lfloor k/2 \rfloor} \binom{k}{i} \\
&> 1 - \left(\frac{1}{4} - \varepsilon_x^2 \right)^{k/2} \cdot 2^k \\
&\geq 1 - (1 - 4\varepsilon_x^2)^{k/2} \geq 1 - (1 - 4\varepsilon^2)^{k/2} \tag{**}
\end{aligned}$$

Ak chceme, aby $\text{Prob}[A_k(x) = F(x)] \geq 1 - \delta$, stačí zvoliť

$$k = \frac{2 \ln \delta}{\ln(1 - 4\varepsilon^2)}$$

Uvedomme si, že keďže ε aj δ sú konštanty, je konštantou aj k . Preto za zníženie pravdepodobnosti chyby na δ "platíme" len konštantným nárastom zložitosti

$$T_{A_k}(n) = O(kT_A(n)) = O(T_A(n))$$

□

2.2.4 Monte Carlo s neohraničenou chybou

Konštantna ε v 2MC algoritmoch vytvárala akúsi "bariéru", ktorá umožňovala efektívne odlíšiť, s veľkou pravdepodobnosťou, korektné odpovede od nekorektných. Ak upustíme od tejto požiadavky, dostaneme triedu MC, pri ktorej to so znižovaním pravdepodobnosti chyby nebude také optimistické.

MC

Definícia 2.5 *Pravdepodobnostný algoritmus A je typu Monte Carlo s neohraničenou chybou (MC, unbounded-error MC), ak*

$$\text{Prob}[A(x) = F(x)] > 1/2$$

Príklad 2.3 *Predstavme si, že máme taký pravdepodobnostný algoritmus A , ktorý má pre vstupné slovo x až $2^{|x|}$ možných behov, pričom korektných je $2^{|x|-1} + 1$, teda "tesná väčšina". Takýto algoritmus je zrejme MC*

$$\text{Prob}[A(x) = F(x)] = \frac{2^{|x|-1} + 1}{2^{|x|}} = \frac{1}{2} + \frac{1}{2^{|x|}} > \frac{1}{2}$$

Pritom hodnota $\frac{1}{2^{|x|}}$ tu hrá úlohu ε_x z 2MC algoritmov.

Zaujímá nás, koľko opakovaní tohto algoritmu potrebujeme, aby sme pri hlasovaní znížovanie chyby nadpolovičnou väčšinou dosiahli pravdepodobnosť chyby nanajvyšš δ .

Využijeme, čo už vieme – ak chceme $\text{Prob}[A_k(x) = F(x)] \geq 1 - (1 - 4\varepsilon^2)^{k/2} > 1 - \delta$, stačí zvoliť

$$\begin{aligned} k = k(x) &\geq \frac{2 \ln \delta}{\ln(1 - 4\varepsilon_x^2)} = \frac{2 \ln \delta}{\ln(1 - 4 \cdot 2^{-2|x|})} \\ &\geq \frac{2 \ln \delta}{-2^{-2|x|}} \quad // 0 < y < 1 \Rightarrow \ln(1 - y) \leq -y \\ &= (-2 \ln \delta) 2^{2|x|} \end{aligned}$$

To ale znamená, že $\boxed{\text{Time}_{A_k}(x) = (-2 \ln \delta) 2^{2|x|} \cdot \text{Time}_A(x)}$

Uvedený príklad je z akéhosi pohľadu "worst-case". Ak by sme mali $\varepsilon_x = \frac{1}{\log|x|}$, nebolo by to s časom $\text{Time}_{A_k}(x)$ také zlé.

Na záver tejto časti ešte jeden konkrétny príklad MC algoritmu.

Príklad 2.4 *Uvažujme opäť porovnávanie databáz. Tentokrát budeme vstup akceptovať vtedy, ak sú databázy rôzne. Náhodný výber využijeme na "uhádnutie" miesta, kde sa databázy líšia. Výsledkom je Algoritmus UMC.*

Algoritmus 8 UMC

vstup: RI: $X = x_1 \dots x_n$
 RII: $Y = y_1 \dots y_n$

výstup: 1 ak $X \neq Y$

RI

- 1: náhodne zvol $j \in \{1, \dots, n\}$
- 2: pošli j, x_j do RII

RII

- 1: **if** $x_j \neq y_j$ **then** "accept" ▷ 100% istota
 - 2: **else**
 - 3: "accept" s pravdepodobnosťou $1/2 - \frac{1}{2n}$
 - 4: "reject" s pravdepodobnosťou $1/2 + \frac{1}{2n}$
-

Celá komunikácia spočíva v prenesení j, x_j , preto prenášame $\lceil \log(n+1) \rceil + 1$ bitov. *počet bitov*

Analýzu chyby rozdelíme na dve časti podľa korektnej odpovede. Najprv označenia. *UMC je MC*

Označme $C_{j,l}$ takú udalosť, že RI zvolilo (v prvej fáze) index j a RII v druhej fáze $C_{j,l}$ povedalo l (1 je accept, 0 reject).

$$\text{Prob}[C_{j,0}] = \frac{1}{n} \left(\frac{1}{2} + \frac{1}{2n} \right)$$

$$Prob[C_{j,1}] = \frac{1}{n} \left(\frac{1}{2} - \frac{1}{2n} \right)$$

A_l Označme A_l množinu tých behov, ktoré dajú odpoveď l

$X = Y$ Ak $X = Y$, tak neexistuje možnosť, aby $x_j \neq y_j$. V tejto situácii RII rozhoduje pravdepodobnostne. Zaujímá nás pravdepodobnosť korektnej odpovede:

$$\mathbf{Prob}[A_0] = \sum_{i=1}^n Prob[C_{i,0}] = \sum_{i=1}^n \frac{1}{n} \left(\frac{1}{2} + \frac{1}{2n} \right) = \frac{1}{2} + \frac{1}{2n} > 1/2$$

$X \neq Y$ Ak $X \neq Y$, potom existuje taký index j , že $x_j \neq y_j$. Budeme predpokladať, že je jediný (čo nezvýši pravdepodobnosť korektnej odpovede). Zaujímá nás pravdepodobnosť $A_1 = C_j \cup \{C_{i,1} \mid 1 \leq i \leq n, i \neq j\}$, pričom C_j označuje udalosť, keď RI zvolilo jediný index, na ktorom sa X, Y líšia.

$$\begin{aligned} \mathbf{Prob}[A_1] &= Prob[C_j] + \sum_{i \neq j} Prob[C_{i,1}] \\ &= \frac{1}{n} + \sum_{i \neq j} \frac{1}{n} \left(\frac{1}{2} - \frac{1}{2n} \right) \\ &= \frac{1}{2n} + \frac{1}{2n} + \sum_{i \neq j} \left(\frac{1}{2n} - \frac{1}{2n^2} \right) \\ &= \frac{1}{2n} + \sum_{i=1}^n \frac{1}{2n} - \sum_{i \neq j} \frac{1}{2n^2} \\ &= \frac{1}{2n} + \frac{1}{2} - \frac{n-1}{2n^2} = \frac{1}{2} + \frac{1}{2n^2} > \frac{1}{2} \end{aligned}$$

□

2.2.5 Pravdepodobnostné algoritmy pre optimalizačné úlohy

V prípade optimalizačných úloh zrejme nezávislé opakovanie behu algoritmu použijeme na získanie riešenia lepšej kvality.

optimalizačný problém

Definícia 2.6 *Optimalizačný problém je $U = (\Sigma_I, \Sigma_O, L, L_I, Sol, cena, ciel)$, kde*

Σ_I je vstupná abeceda

Σ_O je výstupná abeceda

$L \subseteq \Sigma_I^*$ je jazyk prípustných vstupov

$L_I \subseteq L$ je jazyk aktuálnych vstupov

Sol $Sol : L \rightarrow Pot(\Sigma_O) \forall x \in L$ je $Sol(x)$ množina prípustných riešení

cena je účelová funkcia;
 $cena(x, Sol(x)) \in \mathbb{R}$

ciel $ciel \in \{min, max\}$

optimálne riešenie

Prípustné riešenie $y \in Sol(x)$ nazveme optimálnym riešením pre x , ak pre účelovú funkciu platí $cena(x, y) = \text{ciel}\{(x, z), z \in Sol(x)\}$. Ak y je optimálne riešenie pre x a U , tak označíme

$$cena(x, y) = OPT_U(x)$$

konzistentný algoritmus

Hovoríme, že algoritmus A je *konzistentný* pre U , ak $\forall x \in L_I A(x) \in Sol(x)$. Hovoríme, že algoritmus B *rieši* optimalizačný problém U , ak

– B je konzistentný pre U

– $\forall x \in L_I B(x) = OPT_U(x)$

Poznámka Uvedená definícia zachytáva všetky aspekty súvisiace s definíciou optimalizačnej úlohy: vstup je reťazec nad abecedou Σ_I , ale korektný vstup je z jazyka L_I . Analogicky pre výstup. Často budeme používať zjednodušenú formuláciu,

keď problém identifikujeme množinou vstupov I , zobrazením Sol , ktoré vstupu priraduje množinu prípustných riešení, ohodnocovacou funkciou (m , $cena$,..) a cieľom (\max , \min).

Kvalita riešenia sa posudzuje prostredníctvom chyby. Zaujímá nás aproximačný pomer $Ratio_A(x) = \max\{\frac{OPT(x)}{A(x)}, \frac{A(x)}{OPT(x)}\}$. Možné sú dva prístupy – zaujímame sa o očakávanú hodnotu $Ratio_A(x)$, alebo nás zaujíma pravdepodobnosť toho, že bude tento pomer v rozumných hraniciach.

Definícia 2.7 Algoritmus A je pravdepodobnostný $E[\delta]$ -aproximačný, ak

$E[\delta]$ -
aproximácia

- $Prob[A(x) \in Sol(x)] = 1$
- $E[Ratio_A(x)] \leq \delta \forall x \in L$

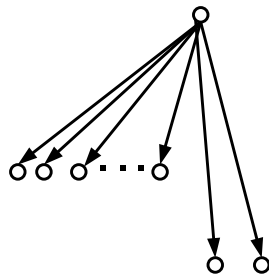
Definícia 2.8 Algoritmus A je pravdepodobnostný δ -aproximačný, ak

δ -aproximácia

- $Prob[A(x) \in Sol(x)] = 1$
- $Prob[Ratio_A(x) \leq \delta] \geq 1/2 \forall x \in L$

Príkladom pravdepodobnostného algoritmu pre optimalizačný problém bol Algoritmus 4 pre problém MAX-SAT: algoritmus vrátil náhodný vektor. Ukázali sme, že ak vstupná formula má m klauzúl, tak náhodný vektor spĺňa $m/2$ klauzúl, čo znamená, že uvedený algoritmus je $E[2]$ -aproximačný.

Hoci uvedené definície 2.7 a 2.8 sú podobné, nie sú, ako vidno na nižšie uvedených príkladoch, totožné.



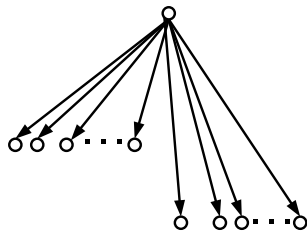
$$Ratio_{A,x}(C_i) = 2 \quad \text{pre } i = 1, \dots, 10$$

$$Ratio_{A,x}(C_i) = 50 \quad \text{pre } i = 11, 12$$

$$E[Ratio_A(x)] = 10$$

$$Prob[Ratio_A(x) \leq 2] = 5/6 \geq 1/2$$

Veľa "dobrých", málo "zlých" spôsobí, že hoci je pravdepodobnosť dobrej odpovede slušná, stredná hodnota je zlá.



$$Ratio_A(x) = 11 \quad \text{pre } 1000 \text{ behov}$$

$$Ratio_A(x) = 1 \quad \text{pre } 999 \text{ behov}$$

$$E[Ratio_A(x)] = \frac{11999}{1999} = 6.0025$$

$$Prob[Ratio_A(x) \leq 10] = \frac{999}{1999} = 0.499.. < 1/2$$

Keď je "dobrých" aj "zlých" skoro rovnako, pravdepodobnosť dobrej odpovede môže byť $< 1/2$, ale stredná hodnota je aj tak dobrá.

Definície sú rôzne, napriek tomu niečo o ich vzťahu povedať môžeme. Vychádzajúc z faktu 2.3 dostávame ako dôsledok Lemu 2.4.

Fakt 2.3 Pravdepodobnosť udalosti, že náhodná premenná x má hodnotu $\leq 2E[x]$, je aspoň $1/2$.

Lema 2.4 Nech $\delta > 0$, U optimalizačný problém a B algoritmus, ktorý ho rieši. Ak algoritmus B je pravdepodobnostný $E[\delta]$ -aproximačný, potom B je 2δ -aproximačný.

Príklad 2.5 Uvažujme problém Max-EkSAT, kde vstupom je formula F v konjunktívnej normálnej forme, pričom každá klauzula má presne k literálov; označme takýto "normálny tvar" k -KNF. Opäť chceme vypočítať vektor, ktorý spĺňa čo najviac klauzúl.

Algoritmus RSAM vráti náhodne zvolený vektor x . Ak bude vstupom formula v tvare k -KNF, tak pre $Z_i(\alpha) = 1 \Leftrightarrow F_i(\alpha) = 1$ máme $E[Z_i] = (1 - 1/2^k)$, z čoho zas pre $Z_F = \sum_{i=1}^m Z_i$ máme $E[Z_F] = \sum E[Z_i] = m(1 - \frac{1}{2^k})$. Keďže $OPT \leq m$,

$$E[\text{Ratio}(F)] = \frac{OPT(F)}{E[Z_F]} \leq \frac{m}{m(1 - \frac{1}{2^k})} = \frac{2^k}{2^k - 1}$$

a teda Algoritmus RSAM je $E[2^k/(2^k - 1)]$ -aproximačný.

Ukážeme, že tento algoritmus je tiež $\frac{2^{k-1}}{2^{k-1}-1}$ -aproximačný. K tomu potrebujeme vyargumentovať, že aspoň polovica náhodných vektorov spĺňa aspoň $m(1 - \frac{1}{2^{k-1}})$ klauzúl.

- $E[Z_F] = m(1 - \frac{1}{2^k})$ znamená, že priemerný počet splnených klauzúl je $m(1 - \frac{1}{2^k})$
- $m(1 - \frac{1}{2^k}) = \frac{m+m(1 - \frac{1}{2^{k-1}})}{2}$
- Ukážeme, že keby viac ako $1/2$ vektorov spĺňala menej ako $m(1 - \frac{1}{2^{k-1}})$ klauzúl, nemohli by sme dostať priemer $m(1 - \frac{1}{2^k})$.

Nech ℓ je počet tých vektorov, ktoré spĺňajú menej ako $m(1 - \frac{1}{2^{k-1}})$ klauzúl; $u = 2^n - \ell$. Potom

$$\begin{aligned} m \cdot \left(1 - \frac{1}{2^k}\right) &= E[Z_F] \leq \frac{1}{2^n} \left(\ell \cdot \left[m \left(1 - \frac{1}{2^{k-1}}\right) - 1 \right] + u \cdot m \right) \\ 2^n m \cdot \left(1 - \frac{1}{2^k}\right) &< \left(\ell m \left[1 - \frac{1}{2^{k-1}}\right] + u \cdot m \right) \end{aligned}$$

z čoho $\ell < \frac{1}{2}2^n$

- Preto aspoň polovica behov musí spĺňať aspoň $m(1 - \frac{1}{2^{k-1}})$ klauzúl.

◇

MaxCut

Príklad 2.6

vstup: neohodnotený graf $G = (V, E)$
výstup: rozklad (V_1, V_2) množiny vrcholov V
cena: $\text{cost}((V_1, V_2)) = |E \cap (V_1 \times V_2)|$
cieľ: max

Ukážeme, že náhodná voľba je $E[2]$ -aproximačný algoritmus. Indikačná premenná bude počítat počet hrán rezu. Nech $e = (u, v)$

$$X_e = \begin{cases} 0, & \text{vrcholy } u, v \text{ nepatria do rezu, ale sú spoločne v jednej z množín } V_1, V_2 \\ 1, & \text{hrana } (u, v) \text{ patrí do rezu} \end{cases}$$

Algoritmus 9 RC

$V_1 = V_2 = \emptyset$
 $\forall v \in V$ náhodne zvol i a $V_i \leftarrow V_i \cup v$
return (V_1, V_2)

$$\begin{aligned} E[X_e] &= \text{Prob}[x_e = 1] \\ \text{Prob}[x_e = 1] &= \text{Prob}[u \in V_1, v \in V_2] + \text{Prob}[u \in V_2, v \in V_1] \\ &= \frac{1}{2} \cdot \frac{1}{2} + \frac{1}{2} \cdot \frac{1}{2} = \frac{1}{2} \end{aligned}$$

Takže

$$E[x_e] = 1/2$$

$$X = \sum_{e \in E} x_e, \quad E[X] = |E|/2$$

$$E[\text{Ratio}] = \frac{OPT}{E[X]} \leq \frac{|E|}{E[X]} = 2$$

◇

Kapitola 3

Pravdepodobnostné zložitosné triedy

¹ Zamyslime sa nad formálnym výpočtovým modelom, ktorý by bol adekvátnym pre definovanie pravdepodobnostných zložitosných tried. Celkom prirodzeným sa javí obohatenie inštrukčnej sady TS o "hádzanie mincou"—pri programovaní celkom bežne využívame *RND*-generovanie náhodných objektov z danej (konečnej) usporiadanej množiny dopredu určenej veľkosti. Ukážeme, že to nie je potrebné—vystačíme s klasickým nedeterministickým Turingovým strojom, ktorému ale upravíme podmienku akceptácie.

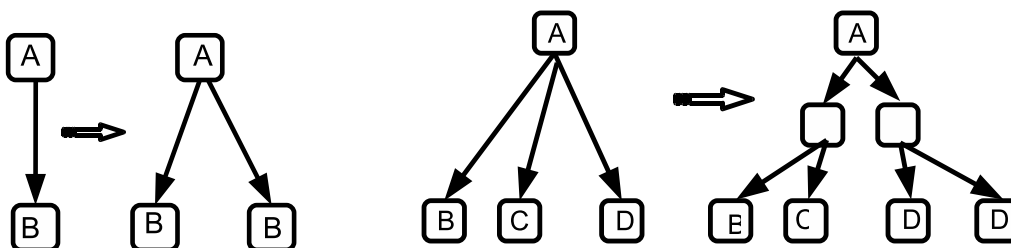
Pre zjednodušenie budeme uvažovať tzv. štandardizovaný TS

Definícia 3.1 (Štandardizovaný TS) Štandardizovaný Turingov stroj je *nedeterministický TS* M , ktorý ma nasledujúce vlastnosti *štandardizovaný Turingov stroj*

1. v každom kroku výpočtu má M možnosť výberu z presne dvoch možností
2. časová zložitosť $t(n)$ stroja M je konštruovateľná
3. každý výpočet stroja M na vstupe w je presne dĺžky $f(|w|)$

Je zrejmé, že k ľubovoľnému NTS M , ktorého časová zložitosť je konštruovateľná, existuje ekvivalentný štandardizovaný TS M' :

- deterministický krok nahradíme dvomi ekvivalentnými krokmi
- výber z k možných krokov nahradíme postupnosťou $\lceil \log k \rceil + 1$ krokov, z ktorých každý vyberá presne z dvoch možností
- pridáme počítanie krokov



Zaujímajú nás efektívne výpočty, obmedzíme sa preto len na polynomiálne algoritmy.

Pravdepodobnostný TS(PTS) dostaneme zo štandardizovaného, keď

PTS

- každý výpočet je dĺžky $p(|x|)$
- v každom kroku robíme výber nasledujúceho kroku pomocou fair-mince. Pri fixovanom PTS je teda konkrétny výpočet určený vstupom x a postupnosťou náhodných bitov $y \in \{0, 1\}^{p(|x|)}$

Klasifikáciu pravdepodobnostných tried dostaneme na základe podmienky akceptovania. Keďže predpokladáme, že každý výpočet na danom vstupe je rovnako dlhý, pravdepodobnosť, resp. početnosť akceptovania/zamietania ľahko určíme spočítaním akceptujúcich/zamietajúcich výpočtov.

NP

Z nejakého pohľadu je aj nedeterministický TS pravdepodobnostný; len s pravdepodobnosťou to nie je najlepšie... Slovo sa akceptuje, keď existuje aspoň jeden akceptujúci výpočet. To možno definovať pomocou PTS M takto:

$$x \in L \iff \text{Prob}(M(x) = 1) > 0$$

3.1 Trieda RP

Definícia 3.2 (Trieda RP) Monte Carlo TS M je štandardizovaný TS polynomiálnej časovej zložitosti $p(n)$, pre ktorý platí

1. $w \in L \Rightarrow \text{Pr}[M(w) = \text{accept}] \geq 1/2$
2. $w \notin L \Rightarrow M(w) = \text{reject}$

trieda RP

Triedu jazykov rozhodovaných Monte Carlo TS označujeme RP(Randomized Polynomial time)

trieda co-RP

Ak stroju dovolíme robiť chyby len v prípade, keď zamietá, dostaneme triedu co-RP.

Uvedomme si, že RP sú také problémy, keď vieme rýchly algoritmus, ktorý sa môže myliť iba v prípade, ak má vstupné slovo akceptovať. Keďže v prípade slova, ktoré do jazyka nepatrí, je každý výpočet zamietajúci, každý akceptujúci výpočet je 100% akceptovaním.

Pravdepodobnosť chyby ε je v tomto modeli nanajvýš $1/2$. Je hodnota $1/2$ podstatná? Ľahko vidno, že nie.

Nahraďme podmienku 1. v definícii 3.2 podmienkou 1':

$$1'. \quad w \in L \Rightarrow \text{Pr}[M(w) = \text{accept}] \geq \varepsilon, \quad 0 < \varepsilon < 1$$

Potom platí:

¹Papadimitriou: Computational Complexity

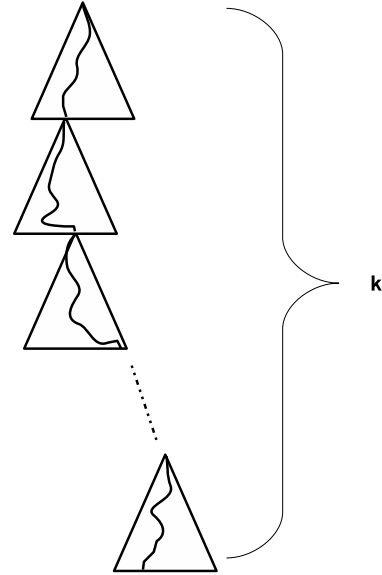
Fakt 3.1 *Ku každému Monte Carlo TS N s podmienkou akceptovania 1' existuje ekvivalentný Monte Carlo M s podmienkou akceptovania 1.*

Dôkaz: Konštrukcia požadovaného stroja je jednoduchá. Stroj M si zapamätá vstup a k -krát po sebe zopakuje výpočet pôvodného stroja N . M akceptuje len vtedy, ak niektorý z k čiastkových výpočtov akceptoval.

Pri slovách, ktoré nie sú z jazyka, sa M nemýli. Aby chybné odpovedal v prípade slova z jazyka, musí všetkých k výpočtov nesprávne skončiť s odpoveďou reject. Preto je chyba stroja M najviac $(1 - \varepsilon)^k$.

Ak chceme dosiahnuť, aby $(1 - \varepsilon)^k \leq \frac{1}{2}$, stačí zvoliť $k \leq \lceil -\frac{1}{\log(1-\varepsilon)} \rceil$.

□



Pravdepodobnosť chyby, ktorú chceme získať, môžeme ohraničiť aj pomocou polynómu. Dostávame alternatívnu charakterizáciu triedy RP.

Veta 3.2 $L \in RP \iff \exists$ polynóm q a PTM M

$$x \in L \implies \text{Prob}[M(x) = 1] > \frac{1}{q(|x|)} \quad x \notin L \implies \text{Prob}[M(x) = 1] = 0$$

Dôkaz:

Ak $L \in RP$, potom stačí zobrať $q(n) = 2$



Majme M z predpokladu vety. Vhodným opakovaním výpočtu M a voľbou kritéria akceptovania dostaneme, čo chceme.



M_t

- t simulácií algoritmu M
- M_t akceptuje ak aspoň jeden z výpočtov $M(x)$ akceptoval

ak $x \notin L$, tak $M(x)$ nemôže akceptovať a $\text{Prob}[M_t(x) = 0] = 1$

$x \notin L$

Podme odhadnúť pravdepodobnosť chyby

$x \in L$

$$\text{Prob}[M_t(x) = 0] = (\text{Prob}[M(x) = 0])^t \leq \left(1 - \frac{1}{q(|x|)}\right)^t$$

Pre dosť veľké t sa pravdepodobnosť chyby dostaneme pod $1/2$. Potrebujeme ale vyargumentovať, že stačí, aby t bolo polynomiálne od $|x|$. Vyplýva to z jednoduchého faktu

$$\lim_{k \rightarrow \infty} \left(1 - \frac{1}{k}\right)^k = e^{-1}$$

□

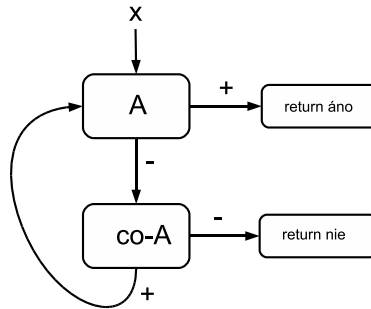
Monte Carlo TS je špeciálnym prípadom polynomiálne ohraničeného nedeterministického TS. Deterministický polytime TS ľahko prerobíme na štandardizovaný TS, ktorý je špeciálnym prípadom Monte Carlo stroja.

Veta 3.3 $P \subseteq RP \subseteq NP$

3.2 Trieda ZPP

Problémy z triedy RP sa mýlia na vstupoch z jazyka, problémy z triedy co-RP zas na vstupoch, ktoré z jazyka nie sú. Žiaden počet opakovaní negarantuje, že máme 100% správnu odpoveď.

Inak je tomu v prípade problémov z triedy $RP \cap co-RP$. Majme dva algoritmy A a co-A, A typu RP a co-A typu co-RP. Pre algoritmus A je *vždy* korektná pozitívna odpoveď, pre co-A zas negatívna. Ak budeme striedavo spúšťať algoritmy A a co-A, po k opakovaníach oboch je pravdepodobnosť toho, že *nemáme* korektnú odpoveď menšia ako 2^{-k} . Máme však garantované, že v konečnom čase korektnú odpoveď dostaneme.



Sme v inej situácii ako doteraz—nevieme síce dopredu, koľko pokusov bude treba, vieme však identifikovať, keď ich je dosť. Algoritmy s uvedenými vlastnosťami sú tzv. Las Vegas algoritmy a príslušnú triedu problémov označujeme ZPP.

ZPP

Definícia 3.3 (Trieda ZPP) $ZPP = RP \cap co-RP$

Veta 3.4 Pre ľubovoľný jazyk L sú nasledovné dve podmienky ekvivalentné

1. $L \in ZPP$
2. existuje štandardizovaný TS M polynomiálnej časovej zložitosti, ktorého každý výpočet končí v stave q_{accept} , q_{reject} alebo q_{stop} . Pritom
 - ak $w \in L$ tak \forall výpočet končí v stave q_{accept} alebo q_{stop}
 - ak $w \notin L$ tak \forall výpočet končí v stave q_{reject} alebo q_{stop}
 - pravdepodobnosť, že výpočet skončí v stave q_{stop} , je nanajvyšš $1/2$

Dôkaz:

$1 \Rightarrow 2$

Požadovaný stroj M získame napr. takto: M simuluje RP stroj; ak RP stroj akceptuje, je to korektná odpoveď, preto aj M akceptuje. Ak RP stroj zamietá, M odsimuluje co-RP stroj; ak tento zamietá, odpoveď je korektná a preto aj M zamietá; v opačnom prípade zastane v stave q_{stop} . Algoritmus nedá korektnú odpoveď len v tom prípade, ak *ani jeden* z RP a co-RP algoritmov nedal definitívnu—korektnú—odpoveď. Preto $Pr[M(w) = q_{stop}] \leq 1/4$.

$2 \Rightarrow 1$

Ľahko vidno, že stav q_{stop} je "záchytným" v situácii, keď si stroj nie je istý. Môžeme ho teda bez následkov zameniť za stav, ktorým stroj "klame".

Ak chceme získať RP stroj, ktorý sa môže myliť keď hovorí nie, stačí v stroji M nahradiť stav q_{stop} stavom q_{reject} . Analogicky pre co-RP stroj, ktorý sa môže myliť pri odpovedi áno; stačí nahradiť stav q_{stop} v M stavom q_{accept} . \square

Polynomiálne Las Vegas algoritmy dávajú korektnú odpoveď, ktorú s veľkou pravdepodobnosťou získame v polynomiálnom čase. Sú teda veľmi užitočné v situáciách, keď potrebujeme *naozaj korektnú* odpoveď.

$$P \subseteq ZPP \subseteq NP \cap \text{co-NP}$$

3.3 Trieda PP

Ak necháme TS "rozhodovať väčšinou", a to aj veľmi tesnou väčšinou, dostaneme triedu PP.

Definícia 3.4 (Trieda PP) *Jazyk L patrí do triedy PP práve vtedy, ak existuje trieda PP polynomiálne časovo ohraničený štandardizovaný TS M , pričom*

$$w \in L \Leftrightarrow \Pr[M(w) = \text{accept}] > 1/2$$

Príkladom jazyka, ktorý patrí do triedy PP, je MAJ-CNF, definovaný nasledovne:

$$\text{MAJ-CNF} = \{F(x_1, \dots, x_n) \mid \exists \text{ aspoň } 2^{n-1} (x_1, \dots, x_n) \in \{0, 1\}^n : F(x_1, \dots, x_n) = 1\}$$

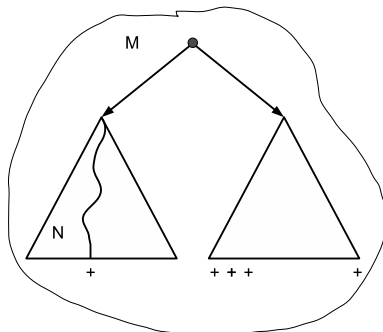
Ako PP stroj stačí zobrať nedeterministický stroj, ktorý háda priradenia $(x_1, \dots, x_n) \in \{0, 1\}^n$ a akceptuje, ak $F(x_1, \dots, x_n) = 1$. Je zrejmé, že $F \in \text{MAJ-CNF}$ práve vtedy, ak viac ako polovica výpočtov akceptuje.

Lema 3.5 $NP \subseteq PP \subseteq PSPACE$

Dôkaz:

Polynomiálne ohraničený TS vieme simulovať v PSPACE. Ak do simulácie pridáme počítanie akceptujúcich konfigurácií, priestorová zložitosť sa nezvýši (dĺžka binárneho zápisu # listov stromu hĺbky h je h)

Nech N je NTS polynomiálnej zložitosti $p(n)$. Aby sme N "prerobili" na PP stroj M , musíme, pre prípad akceptovania, zabezpečiť dostatočné množstvo akceptujúcich konfigurácií. Idea je veľmi jednoduchá—pridáme podstrom rovnakej hĺbky $p(n)$, ktorého všetky listy akceptujú.



Stroj M

1. v počiatočnom stave si nedeterministicky vyberie z dvoch možností q_{0N}, q_M
2. zo stavu q_{0N} začne simulovať N
3. zo stavu q_M vytvára výpočty dĺžky $p(n)$, ktoré všetky akceptujú.

Ak $w \in L(N)$, tak # akceptujúcich výpočtov M je aspoň $1 + 2^{p(n)}$. To znamená, že $Pr[M(w) = q_{accept}] > 1/2$ a stroj M korektne akceptuje. Naopak, ak $w \notin L(N)$, tak # akceptujúcich výpočtov M je presne $2^{p(n)}$, preto $Pr[M(w) = q_{accept}] = 1/2$, čo k akceptovaniu nestačí a stroj M korektne zamietá. \square

Vzhľadom na to, že k akceptovaniu PP stroja stačí, aby počet akceptujúcich výpočtov bol o 2 väčší ako počet zamietajúcich výpočtov, je pravdepodobnosť chyby, ktorej sa môžeme v tomto najhoršom prípade dopustiť, $\frac{1}{2} - \frac{1}{2^{p(n)}}$, kde $p(n)$ je polynóm, ohraničujúci časovú zložitosť PP stroja. V praxi sa teda dosť ťažko rozlišuje, aká vlastne je správna odpoveď.

Sme akoby v situácii, keď máme takú mincu, ktorej jedna strana padá s pravdepodobnosťou $1/2 + \varepsilon$ a druhá s pravdepodobnosťou $1/2 - \varepsilon$ a my potrebujeme zistiť, ktorá strana je ktorá. Koľkokrát treba mincou hodiť, aby sme si odpoveďou boli dostatočne istí? Využijeme nasledujúci fakt.

Chernoff

Fakt 3.6 *Nech x_1, \dots, x_N sú nezávislé náhodné premenné, ktoré nadobúdajú hodnotu 1 resp. 0 s pravdepodobnosťou p , resp. $1 - p$. Nech $X = \sum_{i=1}^N x_i$. Potom pre $0 \leq \Theta < 1$*

$$Pr[X \geq (1 + \Theta)pN] \leq e^{\frac{1}{3}\Theta^2 pN} \quad (3.1)$$

Ak vo vzťahu (3.1) dosadíme $p = 1/2 - \varepsilon$ a $\Theta = \frac{\varepsilon}{\frac{1}{2} - \varepsilon}$, môžeme pravdepodobnosť toho, že $X = \sum_{i=1}^N x_i \geq N/2$ odhadnúť hodnotou $e^{-\frac{1}{6}\varepsilon^2 N}$. To teda znamená, že ak chceme určiť, ktorá strana mince padá s pravdepodobnosťou $\frac{1}{2} + \varepsilon$, stačí spraviť $N = \frac{1}{\varepsilon^2}$ pokusov a zobrať tú, ktorá padla častejšie. Pravdepodobnosť chyby v tomto prípade bude $e^{-\frac{1}{6}}$.

Čo to hovorí o PP stroji? Predpokladajme, že sme PP štandardizovaný stroj spustili opakovane N -krát. Nech $x_i = 1$ práve vtedy, ak odpoveď i -teho behu bola zamietavá. Keďže v najhoršom prípade môže nastať² $\varepsilon = 2^{-p(n)}$, potrebovali by sme $N = \frac{1}{\varepsilon^2} = 2^{2p(n)}$, čo je *exponenciálny počet opakovaní*. To nie je príliš realistické...

3.4 Trieda BPP

Namiesto rozhodovania väčšinou zavedieme rozhodovanie jasnou väčšinou—aspoň tri štvrtiny výpočtov sa musia zhodnúť na výsledku.

Definícia 3.5 (Trieda BPP) *Jazyk L patrí do triedy BPP práve vtedy, ak existuje polytime štandardizovaný TS N taký, že pre každý vstup w*

1. *ak $w \in L$ tak aspoň $3/4$ výpočtov stroja N sú akceptujúce*
2. *ak $w \notin L$ tak aspoň $3/4$ výpočtov stroja N sú zamietajúce*

Trieda BPP zostane nezmenená, ak namiesto $3/4$ vezmeme ľubovoľnú konštantu p , $1/2 < p \leq 1$.

Existujú dôvody sa prikláňať k názoru, že $BPP = P$. Prečo?

- problém testovania prvočíselnosti, ktorý bol dlho kandidátom na separáciu viacerých tried, a ktorý bol ukázkovým príkladom problému z BPP, sa ukázal byť z P
- pravdepodobnosť chyby vieme rozumne znižovať (Veta 3.8)

² p je polynom ohraničujúci čas PP stroja

- problémy z BPP majú polynomiálne obvody (Veta 3.8), preto je nepravdepodobné, aby $NP \subseteq BPP$.

Veta 3.7 $L \in BPP$ vtedy a len vtedy, ak pre každý polynóm p existuje štandardizovaný TS rozhodujúci L s polynomiálnou časovou zložitou a s pravdepodobnosťou chyby nanajvyš $(\frac{1}{2})^{p(n)}$

Dôkaz: Ak štandardizovaný TS má pravdepodobnosť chyby $\leq (\frac{1}{2})^{p(n)}$, tak spĺňa $\boxed{\Leftarrow}$ definíciu BPP stroja.

Predpokladajme, že $L \in BPP$ a M je ten štandardizovaný stroj, ktorý akceptuje L s pravdepodobnosťou chyby $\varepsilon < 1/2$. Konštrukcia požadovaného stroja N je založená na vhodnom počte nezávislých opakovaní výpočtov stroja M . $\boxed{\Rightarrow}$

Označme $\delta = (1 - \varepsilon)$ pravdepodobnosť toho, že výpočet je správny. K danému polynómu $p(n)$ vezmeme polynóm $q(n)$ tak, aby

$$\frac{1}{2}(2q(n) + 1) > c \cdot p(n), \quad (4\varepsilon\delta)^c < \frac{1}{2}$$

Vhodnosť takejto voľby bude zrejmá neskôr...

Stroj N

popis N

1. vypočíta $m \leftarrow 2q(n) + 1$ a nastaví si počítadlo akceptujúcich výpočtov $c_{accept} \leftarrow 0$
2. $2q(n) + 1$ krát simuluje výpočet M (w), pričom si v c_{accept} udržiava aktuálny počet akceptujúcich výpočtov
3. ak po skončení je $c_{accept} > q(n)$, N akceptuje, inak M zamietá

Je zrejmé, že časová zložitnosť popísaného stroja N je polynomiálna. Ostáva ukázať, ako je to s pravdepodobnosťou chyby.

Pravdepodobnosť toho, že práve j výpočtov dá správnu odpoveď, je $\binom{m}{j} \delta^j \varepsilon^{m-j}$. Pravdepodobnosť toho, že stroj *nedá* správnu odpoveď, je rovná pravdepodobnosti, že ju dá nanajvyš $q(n)$ výpočtov. Preto

$$\begin{aligned} Pr[\text{chyba}] &= \sum_{j=0}^{q(n)} \binom{m}{j} \delta^j \varepsilon^{m-j} \\ &\leq \delta^{m/2} \cdot \varepsilon^{m/2} \cdot \sum_{j=0}^{q(n)} \binom{m}{j} \quad // \delta > \varepsilon \Rightarrow \delta^j \varepsilon^{m-j} \leq \delta^{m/2} \cdot \varepsilon^{m/2} \\ &\leq \delta^{m/2} \cdot \varepsilon^{m/2} \cdot 2^m \\ &= (4 \cdot \varepsilon \cdot \delta)^{m/2} \end{aligned}$$

Vzhľadom k voľbe c , $q(n)$ dostávame

$$Pr[\text{chyba}] \leq (4\varepsilon\delta)^{m/2} \leq \left(\frac{1}{2}\right)^{m/2c} \leq \left(\frac{1}{2}\right)^{p(n)}$$

◇

Dôkaz môžeme spraviť využitím toho, čo už vieme. V časti 2.2.3 sme pre algoritmy s ohraničenou chybou ukázali, že ak opakujeme BPP algoritmus t krát a rozhodneme väčšinou (Algoritmus A_t),

$$Pr[A_t(x) = F(x)] \geq 1 - (1 - 4\varepsilon^2)^{t/2}$$

Pritom pre pôvodný algoritmus A platilo, že $Pr[A(x) = F(x)] \geq 1/2 + \varepsilon$. V prípade BPP je $\varepsilon = 1/4$. Chceme teda určiť t tak, aby

$$\left(\frac{1}{2}\right)^{p(n)} \geq (1 - 4\varepsilon^2)^{t/2} = \left(\frac{3}{4}\right)^{t/2}$$

$$t \geq \frac{2p(n)}{\log 4/3}$$

□

A teraz ukážeme vzťah BPP a polynomiálnych obvodov³.

BPP \rightarrow obvod

Veta 3.8 Ak $L \in BPP$, tak L má polynomiálny obvod.

Dôkaz: Skôr, ako prejdeme k dôkazu, si uvedomme, že keby ten polynomiálny obvod bol uniformný, tak vzhľadom na vzťah⁴ uniformných polynomiálnych obvodov a triedy P by sme mali, že $BPP = P$; obvod nebude uniformný, jeho konštrukcia nie je efektívna.

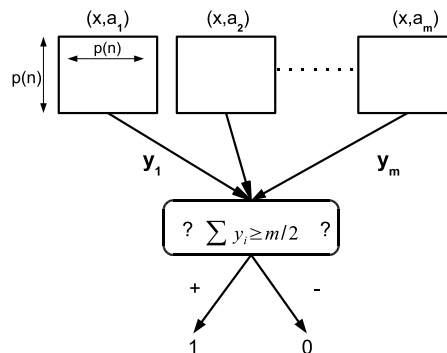
$L \in BPP$ znamená, že existuje PTS M rozhodujúci jasnou väčšinou—aspoň $3/4$ výpočtov hovoria korektne. Nech $p(n)$ je časová zložitosť M . Ak by sme PTS M pridali postupnosť α náhodných bitov popisujúcu výsledok hádzania mincou pri výpočte na slove x , popíšeme konkrétny—deterministický—výpočet $M(x, \alpha)$ tohto stroja. Ak nás tento výpočet dovedie k správnej odpovedi, bola voľba α "vhodná". Dôkaz spočíva v argumentácii o existencii takej malej množiny takýchto postupností, ktorá je "vhodná" pre všetky vstupy rovnakej dĺžky.

Uvažujme

- postupnosť náhodných bitov $A_n = (a_1, \dots, a_m)$, $m = 12(n+1)$, $a_i \in \{0, 1\}^{p(n)}$
- vstup x dĺžky n
- obvod C , ktorý paralelne simuluje $M(x, a_1), \dots, M(x, a_m)$ a potom rozhodne väčšinou.

Lema 3.9 poskytuje korektnosť našej konštrukcie.

Polynomialita C vyplýva zo simulácie TS na obvodoch; pre $T(n)$ -časovo ohraničený TS je simulujúci obvod veľkosti $O(T(n))$.



Lema 3.9 $\forall n > 0 \exists A_n$, množina $m = 12(n+1)$ $p(n)$ -bitových reťazcov takých, že $\forall x \in \{0, 1\}^n$ menej ako $1/2$ a_i vedie k nekorektnej odpovedi $M(x, a_i)$.

Dôkaz: Potrebujeme ohraničiť pravdepodobnosť toho, že $\forall x \in \{0, 1\}^n$ je viac ako polovica a_i v A_n "dobrá". Vzhľadom k tomu, že stroj M je BPP, $\forall x \in \{0, 1\}^n$ je

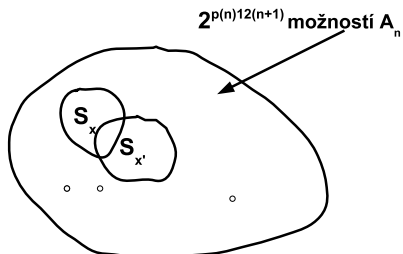
³Kapitolka o obvodoch sa nachádza v appendixe

⁴ $L \in P \Leftrightarrow L$ má uniformný obvod polynomiálnej veľkosti.

nanajvýš $1/4$ z a_i "zlá". Volili sme nezávisle, preto očakávaný počet zlých volieb je $m/4$. Využitím Chernoffovho odhadu dostávame (pre všetky x)

$$Pr[\# \text{ zlých} \geq m/2] \leq e^{-\frac{m}{12}} < \frac{1}{2^{n+1}}$$

Pravdepodobnosť toho, že *existuje* vstup x , ktorý *nemá* v A_n akceptujúce a_i , je nanajvýš $2^n \cdot \frac{1}{2^{n+1}} = 1/2$. Preto má náhodne zvolené A_n s pravdepodobnosťou aspoň $1/2$ požadované vlastnosti. \square



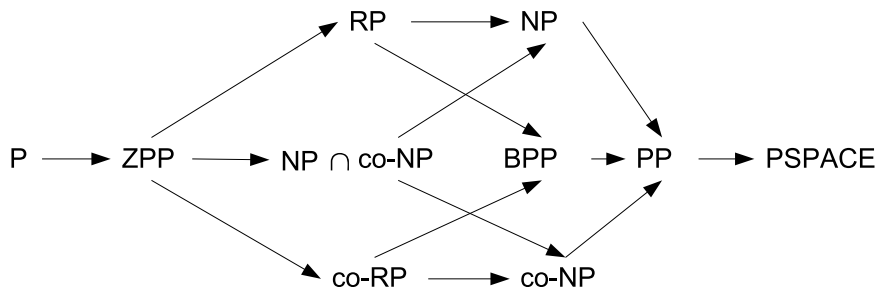
- S_x je množina tých, ktoré nedajú pre x korektnú odpoveď podľa väčšiny;
- $|S_x| \leq \frac{2^{12(n+1)p(n)}}{2^{n+1}}$
- $\sum_x |S_x| \leq 2^n \cdot \frac{2^{12p(n)(n+1)}}{2^{n+1}} < 2^{12p(n)(n+1)}$

Máme teda možnosť pre "dobré" A_n , aj keď ho efektívne (v polynomiálnom čase) nevieme hľadať. \square

Zrejme

$$RP \subseteq BPP \subseteq PP$$

Prehľad vzťahu medzi jednotlivými triedami poskytuje nasledujúca schéma; znakom \rightarrow je znázornená inklúzia.



3.5 Zdroje náhodných postupností/bitov

Triedy RP a BPP sú zaujímavé triedy problémov, ktoré však vyžadujú existenciu dobrého zdroja náhodných čísel. Čo ale považujeme za dobrý zdroj náhodných čísel?

Definícia 3.6 Perfektný zdroj náhodných bitov je náhodná premenná, ktorej hodnotami sú nekonečné postupnosti $x_1, x_2, \dots \in \{0, 1\}^*$ bitov také, že *perfect random source*

$$\forall (y_1, \dots, y_n) \in \{0, 1\}^n Pr[x_i = y_i, i = 1, \dots, n] = 2^{-n}$$

Inými slovami je to taká nekonečná postupnosť, kde každý prvok x_i možno považovať za výsledok nezávislého pokusu, pričom $Pr[x_i = 1] = Pr[x_i = 0] = 1/2$. Vyžadujeme teda

- nezávislosť—pravdepodobnosť $x_i = 1$ nezávisí od x_1, \dots, x_{i-1} ; výsledok i-teho hodu mincou nezávisí od výsledkov predchádzajúcich hodov
- "fairness"—minca musí byť korektná; pravdepodobnosť $x_i = 1$ musí byť *presne* $1/2$

Podstatná je nezávislosť pokusov, pretože korektnosť vieme obísť. Uvažujme postupnosť $X = x_1, \dots$ bitov, ktorá je výsledkom hádzania nekorektnou mincou. Pomocou tejto postupnosti vieme vyrobiť náhodnú postupnosť, v ktorej $Pr[x_i = 0] = Pr[x_i = 1] = 1/2$, takto:

- rozdelíme postupnosť $X = x_1x_2x_3x_4\dots$ na bloky dĺžky 2; $X = (x_1, x_2), (x_3, x_4), \dots$
- vypustíme bloky 00, 11
- vo vzniknutej postupnosti nahradíme $01 \rightsquigarrow 0$, $10 \rightsquigarrow 1$

Nepotrebujeme vedieť pravdepodobnosť p , stačí, aby $0 < p < 1$ bola nemenná. Koľko pokusov nekorektnou mincou stačí, aby sme dostali "korektnú" postupnosť dĺžky n ? Očakávaná dĺžka je $\frac{2}{1-c}$, kde $c = p^2 + (1-p)^2$ je tzv. *coincidence probability*.

Zdá sa, že fyzikálne zdroje majú problém s nezávislosťou. Čo sa stane, ak od nej upustíme?

δ -random source **Definícia 3.7** *Nech $0 < \delta \leq 1/2$ a funkcia $p : \{0, 1\}^* \rightarrow (\delta, 1 - \delta)$. δ -náhodný zdroj S_p je náhodná premenná-nekonečná postupnosť náhodných bitov, pričom pravdepodobnosť, že prvých n bitov má špecifické hodnoty y_1, \dots, y_n , je*

$$\prod_{i=1}^n (y_i p(y_1, \dots, y_{i-1}) + (1 - y_i)(1 - p(y_1, \dots, y_{i-1})))$$

Pravdepodobnosť, že i -ty bit=1 je $p(y_1, \dots, y_{i-1})$, čo je číslo medzi δ a $1 - \delta$, *ľubovoľne* závislé od prvých $(i - 1)$ hodnôt.

$\frac{1}{2}$ -náhodný zdroj je perfektný zdroj, pre $\delta < 1/2$ hovoríme o slightly random zdroji. Takéto slightly random fyzikálne zdroje existujú, zdá sa však, že pre pravdepodobnostné algoritmy nie sú (priamo) vhodné. Ak máme MC algoritmus a $\delta \lll 1/2$, potom algoritmus môže príliš často klamať.

2SAT-random walk

Príklad 3.1 *Uvažujme splniteľnosť 2SAT—pre vstupnú formulu $F(x_1, \dots, x_n)$ v 2SAT (každá klauzula má presne 2 členy) sa pýtame, či je splniteľná. Algoritmus je jednoduchý—riešenie hľadáme náhodnou prechádzkou po priestore potenciálnych riešení.*

Algoritmus 10 2SAT-random walk

- 1: zvol priradenie α náhodne
 - 2: **if** $F(\alpha) = 1$ **then** return("splniteľná")
 - 3: **else if** ani jedna klauzula v $F(\alpha)$ nie je splnená **then** return("asi nesplniteľná")
 - 4: **else**
 - 5: v nesplnenej klauzule náhodne zmeň priradenie jedného z literálov $0 \leftrightarrow 1$
 - 6: opakuuj nanajvyš r krát
 - 7: ak ani po r "zmenách" priradenie nie je spĺňajúce, return("asi nesplniteľná")
-

Pri analýze algoritmu využijeme nasledujúcu lemu:

Lema 3.10 *Nech X je náhodná premenná s hodnotami z \mathbb{N} , potom $\forall k > 0$*

$$Pr[X \geq kE[X]] \leq 1/k$$

Dôkaz: Nech p_i označuje pravdepodobnosť, že $x = i$. potom

$$E[X] = \sum_i ip_i = \sum_{i < kE[X]} ip_i + \sum_{i \geq kE[X]} ip_i \geq kE[X] Pr[x \geq kE[X]]$$

$$\frac{1}{k} \geq Pr[X \geq kE[X]]$$

□

Lema 3.11 Ak vstupná formula F je splniteľná, potom pre $r = 2n^2$ je pravdepodobnosť, že Algoritmom 3.1 nájdeme spĺňajúce priradenie, aspoň $1/2$.

Dôkaz: Ukážeme, že stredná hodnota očakávaného počtu výmen/preklopení, keď sa počítaťné priradenie α líši od spĺňajúceho v i bitoch, je n^2 . Na základe lemy 3.10 to potom znamená, že ak je formula splniteľná, $2n^2$ opakovaní stačí, aby sme spĺňajúce priradenie našli s pravdepodobnosťou aspoň $1/2$.

Predpokladajme, že formula je splniteľná a kvôli jednoduchosťi fixnime jedno spĺňajúce priradenie \hat{T} . Označme $t(i)$ —očakávaný počet výmen/preklopení v prípade, keď začiatkové priradenie α sa od \hat{T} líši v i bitoch.

- $t(0) = 0$; uvedomme si, že 0 môžeme dostať aj vtedy, ak na začiatku sa α líši od \hat{T} —formula môže mať viacero spĺňajúcich priradení
- zmena priradenia hodnoty literálu v klauzule $x \vee y$ spôsobila, že v \hat{T} je aspoň jeden z literálov x, y true;

$$t(i) \leq \frac{1}{2} (t(i-1) + t(i+1)) + 1$$

$$t(n) \leq t(n-1) + 1$$

- nahradíme nerovnosti rovnosťami, čo odpovedá tomu, že zabudneme, že sú aj iné spĺňajúce priradenia a že sa T a \hat{T} môžu líšiť v oboch literáloch—nie jeden, ale oba literály v $x \vee y$ majú byť true⁵

$$\left. \begin{array}{l} x(0) = 0 \\ x(i) = \frac{1}{2} (x(i-1) + x(i+1)) + 1 \\ x(n) = x(n-1) + 1 \end{array} \right\} x(i) \geq t(i)$$

$$\left. \begin{array}{l} \sum_{i=0}^n x(i) = \frac{1}{2} (x(0) + x(2)) + 1 \\ \quad + \frac{1}{2} (x(1) + x(3)) + 1 \\ \quad + \frac{1}{2} (x(2) + x(4)) + 1 \\ \quad \vdots \\ \quad + \frac{1}{2} \left(x(n-2) + \underbrace{x(n)}_{x(n-1)+1} \right) + 1 \\ \quad + x(n) \\ = \frac{1}{2} x(1) + \sum_{i=2}^{n-1} x(i) + (n-1) + 1/2 \\ x(0) + x(1) + x(n) = \frac{1}{2} x(1) + (n-1) + 1/2 \end{array} \right\} \begin{array}{l} x(1) = 2n-1 \\ x(2) = 4n-4 \\ \vdots \\ x(i) = 2in - i^2 \\ x(n) = n^2 \end{array}$$

- $t(i) \leq x(i) \leq x(n) = n^2$, preto očakávaný počet krokov je $\leq n^2$

⁵uvedomme si, že to len zvýši $t(i)$

Keďže očakávaný počet výmen/preklopení pre nájdenie \hat{T} je nanajvýš n^2 , tak

$$Pr[i \geq 2n^2] \leq 1/2$$

Ak F je splniteľná, tak s pravdepodobnosťou väčšou ako $1/2$ nájdeme \hat{T} počas $2n^2$ krokov. \square

Nech by 2SAT malo jediné spĺňajúce priradenie \hat{T} . Definujme p tak, že S_p spôsobí, že prekopíme vždy ten literál, ktorý súhlasí s \hat{T} , s pravdepodobnosťou $1 - \delta$

- zoberme δ -náhodný zdroj S_p s $\delta \leq 1/2$, potom (možno) potrebujeme exponenciálne veľa krokov, aby sme našli \hat{T}
- keby $p' = 1/2$, bolo by to fajn
- slabo náhodný zdroj S_p je akoby protivník, ktorý minimalizuje náš úspech

význam
 δ -random source

Napriek tomu, že δ -náhodný zdroj nemôžeme použiť ako zdroj náhodných bitov, môžeme ho použiť na simuláciu pravdepodobnostného algoritmu s iba polynomiálnym spomalením.

Definícia 3.8 *Nech N je standardizovaný TS, jednotlivé volby sú 0-syn, 1-syn. Výpočet je úplný binárny strom $N(x)$ hĺbky $n := p(|x|)$.*

- *Nech $0 < \delta < 1/2$. δ -priradenie F je zobrazenie z množiny hrán $N(x)$ do intervalu $(\delta, 1 - \delta)$ také, že pre každý vrchol je $F(0\text{-syn}) + F(1\text{-syn}) = 1$ (randomizovaný algoritmus s δ -náhodným zdrojom)*
- *ℓ je list; $Pr[\ell] = \prod_{a \in P(\ell)} F(a)$, kde $P(\ell)$ označuje cestu z koreňa do listu ℓ*
- $Pr[M(x) = " + " | F] = \sum_{\substack{\ell \text{ je list} \\ s \text{ odpoveďou +}}} Pr[\ell]$

δ -RP

Jazyk $L \in \delta$ -RP, ak existuje vyššie popísaný pravdepodobnostný stroj M s δ -náhodným zdrojom tak, že $\forall \delta$ - priradenie F

$$x \in L \Rightarrow Pr[M(x) = " + " | F] \geq 1/2$$

$$x \notin L \Rightarrow Pr[M(x) = " + " | F] = 0$$

δ -BPP

Jazyk $L \in \delta$ -BPP, ak existuje vyššie popísaný pravdepodobnostný stroj M s δ -náhodným zdrojom tak, že $\forall \delta$ - priradenie F

$$x \in L \Rightarrow Pr[M(x) = " + " | F] \geq 3/4$$

$$x \notin L \Rightarrow Pr[M(x) = " - " | F] \geq 3/4$$

Zrejme

- 0 -RP = 0 -BPP = P
pravdepodobnosť listu môže byť 1, preto všetky listy musia hovoriť rovnako
- $1/2$ -RP = RP $1/2$ -BPP = BPP
každá hrana má pravdepodobnosť $1/2$

Čo ak $0 < \delta < 1/2$? Ukážeme, že $\delta - \text{BPP} = \text{BPP}$.

$\delta - \text{BPP} \subseteq \text{BPP}$ Zrejme $\delta - \text{BPP} \subseteq \text{BPP}$.

$\text{BPP} \subseteq \delta - \text{BPP}$ Budeme predpokladať (prečo môžeme?), že pravdepodobnosť chyby BPP stroja N je nanajvyš $1/32$. Nech x je vstup a $p(|x|)$ je hĺbka $N(x)$.

$$\bullet n = p(|x|) \quad k = \left\lceil \frac{3 \log n + 5}{2\delta - 2\delta^2} \right\rceil$$

• **blok** je postupnosť k bitov, ktoré vnímame ako zápis binárneho čísla

• **inner product**

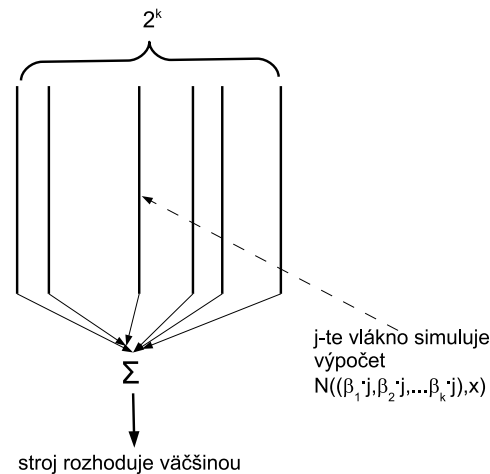
$$\text{nech } \kappa = (\kappa_1, \dots, \kappa_k), \lambda = (\lambda_1, \dots, \lambda_k), \kappa \cdot \lambda = \sum_{i=1}^k \kappa_i \lambda_i \pmod{2}$$

• β , resp. β_i bude označovať k -bitový reťazec, ktorý je výstupom δ -náhodného zdroja S_p . Nemôžeme to používať priamo ako perfektné náhodný reťazec, ale využijeme ho na "oklamanie nepriateľa".

• Idea simulácie je nasledovná: Budeme simulovať 2^k výpočtov $N(x)$, pričom rozhodovanie budeme robiť deterministicky, podľa predvypočítaných "pseudonáhodných" reťazcov. Na záver rozhodneme väčšinou.

• Nech β_j je k -bitový výstup δ -náhodného zdroja S_p , j nech je binárne zapísané reťazcom dĺžky k . Potom i -ty krok j -teho výpočtu sa rozhoduje podľa $\beta_i \cdot j$

• čas $O(n \cdot 2^k) = O(h(|x|))$, kde h je polynóm.



Podme odhadnúť pravdepodobnosť chybnjej odpovede popísaného stroja N_δ . Postupnosť bitov, ktorá vedie ku korektnej odpovedi, je "dobrá"; keď vedie k nesprávnej odpovedi, je "zlá". Označme množinu "zlých" postupností \mathcal{B} . Aby sme ohraničili pravdepodobnosť chyby, potrebujeme odhadnúť, koľko z nami použitých 2^k postupností vedie k nesprávnej odpovedi.

odhad chyby

Podľa nášho predpokladu o chybe BPP je $|\mathcal{B}| \leq \frac{1}{32} 2^k$

\mathcal{B}

Vo výpočte používame postupnosti bitov

$$T = \{\beta_1 \cdot Z, \dots, \beta_n \cdot Z, Z = 0, 1, \dots, 2^k - 1\}$$

Pravdepodobnosť chyby N_δ je $Pr[|T \cap \mathcal{B}| \geq 1|T|/2]$.

Veta 3.12 $Pr[|T \cap \mathcal{B}| \geq 1|T|/2] < 1/4$

Dôkaz: Definujme

$$\text{bias}(\beta_i \cdot Z) = (Pr[\beta_i \cdot Z = 1] - Pr[\beta_i \cdot Z = 0])^2$$

potrebujeme, aby dostatočne veľa bitov ostalo nevychýlených. Ohraničíme priemerný bias

- najskôr ukážeme súvis medzi bias a coincidence probability $\stackrel{def.}{=} \sum_{\beta=0}^{2^k-1} p[\beta]^2$,
kde $p[\beta]$ je pravdepodobnosť, že δ -náhodný zdroj S_p vygeneruje práve β .
- potom ohraničíme tú pravdepodobnosť

Lema 3.13 $\frac{1}{2^k} \sum_{Z=0}^{2^k-1} (Pr[\beta_i \cdot Z = 1] - Pr[\beta_i \cdot Z = 0])^2 = \sum_{\beta=0}^{2^k-1} p[\beta]^2$

Dôkaz: Využijeme, že $Pr[\beta \cdot Z = 0] - Pr[\beta \cdot Z = 1] = \sum_{\beta=0}^{2^k-1} (-1)^{\beta \cdot Z} p[\beta]$. Preto

$$\begin{aligned} \sum_{Z=0}^{2^k-1} (Pr[\beta_i \cdot Z = 1] - Pr[\beta_i \cdot Z = 0])^2 &= \sum_{Z=0}^{2^k-1} \left(\sum_{\beta=0}^{2^k-1} (-1)^{\beta \cdot Z} p[\beta] \right)^2 = \\ &= \sum_{Z=0}^{2^k-1} \sum_{\beta=0}^{2^k-1} p[\beta]^2 + 2 \sum_{Z=0}^{2^k-1} \sum_{\beta_1, \beta_2=0}^{2^k-1} (-1)^{(\beta_1 + \beta_2) \cdot Z} p[\beta_1] p[\beta_2] = \\ &= \sum_{Z=0}^{2^k-1} \sum_{\beta=0}^{2^k-1} p[\beta]^2 + 2 \sum_{\beta_1, \beta_2=0}^{2^k-1} p[\beta_1] p[\beta_2] \underbrace{\sum_{Z=0}^{2^k-1} (-1)^{(\beta_1 + \beta_2) \cdot Z}}_0 = 2^k \sum_{\beta=0}^{2^k-1} p[\beta]^2 \end{aligned}$$

◇

Ohraničenie coincidence probability bloku:

Lema 3.14 Ak β je blok (dĺžky k), ktorý je výstupom δ -náhodného zdroja S_p , potom

$$\sum_{\beta=0}^{2^k-1} p[\beta]^2 \leq (\delta^2 - (1 - \delta)^2)^k$$

Dôkaz: Majme blok $\beta = (x_1, \dots, x_k)$ s pravdepodobnosťou p_i vygenerovania $x_i = 1$. Uvažujme taký blok β' , ktorý sa od β líši len v tom, že $x_i = 1$ ale $x'_i = 0$. Keď rozdelíme sumu na dve časti podľa x_i , dostaneme výraz tvaru

$$Ap_i^2 + A(1 - p_i)^2$$

Tento výraz dosahuje maximum, ak sa p_i a $(1 - p_i)$ líšia čo najviac; položíme teda $p_i = \delta$.

$$\sum_{\beta=0}^{2^k-1} p[\beta]^2 \leq \sum_{i=0}^k \binom{k}{i} \delta^{2i} (1 - \delta)^{2(k-i)} = (\delta^2 + (1 - \delta)^2)^k$$

◇

biased/unbiased Na základe lemm 3.13 a 3.14
bits

- maximálne vychýlenie(bias) bitov j -tého poschodia je $2^k(\delta^2 + (1 - \delta)^2)^k$. Bit $\beta_j \cdot Z$ nazveme *biased(vychýlený)*, ak $bias(\beta_j \cdot Z) \geq 1/n^2$; inak je bit *nevychýlený(unbiased)*.
- ľahko vidno, že nevychýlený bit má pravdepodobnosť toho, že je 1, medzi $\frac{1}{2} - \frac{1}{2n}$ a $\frac{1}{2} + \frac{1}{2n}$.
- počet vychýlených bitov j -tého poschodia je nanajvyš $n^2 2^k (\delta^2 + (1 - \delta)^2)^k$

- celkov počet vychýlených bitov je nanjvyšš $n^3 2^k (\delta^2 + (1 - \delta)^2)^k$;

$$\begin{aligned} n^3 2^k (\delta^2 + (1 - \delta)^2)^k &= 2^k n^3 (1 - 2\delta + 2\delta^2)^{\frac{3 \log n + 5}{2\delta - 2\delta^2}} = \\ 2^k n^3 2^{\log(1 - 2\delta + 2\delta^2) \frac{3 \log n + 5}{2\delta - 2\delta^2}} &\leq 6 2^k n^3 2^{-3 \log n - 5} = 2^k n^{-5} = \mathbf{2^k} \cdot \frac{\mathbf{1}}{\mathbf{32}} \end{aligned}$$

- Uvažujme postupnosť $\beta_1 \cdot Z, \dots, \beta_n \cdot Z \in T$. Nazveme ju vychýlenou, ak *vychýlená* *po-* obsahuje aspoň jeden vychýlený bit. Množina T sa rozpadne na dve $T =$ *stupnosť* (*biaso-* $B \cup U$, pričom B obsahuje vychýlené (biasované) postupnosti a U nevychýle- *vaná*) *né* (unbiased). $B \leq n^3 2^k (\delta^2 + (1 - \delta)^2)^k \leq 2^k / 32$

$$\begin{aligned} E[|B \cap T|] &= \sum_{t_1 \dots t_n \in T} \sum_{b_1 \dots b_n \in B} \prod_{i=1}^n Pr[b_i = t_i] \\ &\leq 2^k / 32 + \sum_{t_1 \dots t_n \in U} \sum_{b_1 \dots b_n \in B} \prod_{i=1}^n Pr[b_i = t + i] \\ &\leq 2^k / 32 + \sum_{t_1 \dots t_n \in U} \sum_{b_1 \dots b_n \in B} \underbrace{\left(\frac{1}{2} + \frac{1}{2n} \right)^2}_{e/2^n} \\ &\leq 2^k / 32 + 2^n \cdot 2^k / 32 e 2^{-n} = \frac{2^k}{32} (1 + e) < \frac{1}{8} 2^k = |T| / 8 \end{aligned}$$

Máme teda $E[|B \cap T|] \leq 1/8$, preto podľa lemy 3.10 $Pr[|T \cap B| \geq 4 \cdot \frac{|T|}{8}] \leq 1/4$

□

Dôsledok 3.15 $\forall \delta > 0 \quad RP = \delta - RP$

⁶pre $0 < \varepsilon < 1$ je $\log(1 - \varepsilon) \leq -\varepsilon$

Kapitola 4

Metódy tvorby pravdepodobnostných algoritmov

Analýzou existujúcich pravdepodobnostných algoritmov možno "vytiahnuť" niekoľko metód, ktoré sa pri ich návrhu využívajú. Často ide o kombináciu, niekedy možno algoritmus vnímať optikou viacerých metód. Začneme krátkym súhrnom metód, ktoré potom podrobnejšie rozvineme a odilustrujeme na príkladoch v nasledujúcich podkapitolách.

Eliminácia protihráča je v podstate metódou vyhýbania sa najhorším prípadom.

Eliminácia protihráča

Keď máme deterministický algoritmus, vstup pre najhorší prípad pre tento konkrétny algoritmus sa väčšinou konštruuje pomerne jednoducho. Pravdepodobnostný algoritmus je vlastne množinou deterministických algoritmov (keď k vstupu pridáme postupnosť náhodných bitov). Keď teda chcem zlý vstup, mal by byť zlým pre dostatočne veľa týchto algoritmov. Dá sa očakávať, že toto nie je vždy možné.

S takýmto prístupom sme sa už stretli.

V prípade zisťovania rovnosti dvoch vzdialených databáz (reprezentovaných binárnymi reťazcami) bola množina deterministických algoritmov určená množinou prvočísel: porovnanie $x \stackrel{?}{=} y$ sme nahradili porovnaním $x \bmod p \stackrel{?}{=} y \bmod p$, kde $p \in Prim(n^2)$. Vieme, že aspoň $\frac{Prim(n^2) - (n - 1)}{Prim(n^2)} = 1 - \frac{2 \ln n}{n}$ -tina odpovedí je korektná.

$R_I, R_{II} \quad x \stackrel{?}{=} y$

↪ náhodná voľba prvočísla vedie k veľkej pravdepodobnosti korektnej odpovede; algoritmus je korektný na takmer každom vstupe

Algoritmus QUICKSORT je príkladom, keď viaceré deterministické stratégie sa líšia voľbou pivota, pričom

QUICKSORT

- každá dáva korektnú odpoveď
- každá je efektívna na väčšine vstupov

$$T(1) = 0, \quad T(n) = n - 1 + T(A_{<}) + T(A_{>})$$

Videli sme, že náhodná voľba pivota viedla v očakávanom prípade k zložitosti rádovo rovnej dolnému odhadu— $O(n \log n)$

Metóda svedkov je vhodná pre také rozhodovacie problémy, keď nás zaujíma, či daný vstup má alebo nemá nejakú vlastnosť V (napr. či to je prvočíslo).

Metóda svedkov

K použitiu metódy je dôležité, aby sme mali

- vhodnú definíciu toho, čo je to svedok. Je to nejaká dodatočná informácia W , ktorá pomáha určiť, či skúmaná vlastnosť platí alebo nie
 $V \rightsquigarrow$ prvočíselnosť $W \rightsquigarrow$ deliteľ
- dostatočne veľkú efektívne konštruovateľnú množinu kandidátov, medzi ktorými je –v prípade existencie– dostatočne veľa svedkov

Pri prvočíselnosti sú kandidátmi $p \in Prim(n^2)$; je ich aspoň $Prim(n^2) - (n - 1)$. Pravdepodobnosť, že náhodne zvolený kandidát je svedok je

$$\frac{\frac{n^2}{\ln n^2} - (n - 1)}{\frac{n^2}{\ln n^2}} \geq 1 - \frac{\ln n^2}{n}$$

Nemáme algoritmus, ktorý efektívne hľadá svedkov. Zoberme prehľadávanie od najmenšieho prvočísla. Keďže "zlých" prvočísel je najvyšš $n - 1$, po n pokusoch určite skončíme. Zložitosť takéhoto prístupu je však $4n \log n$, čo je veľa. Prečo nevieme garantovať nič lepšie? Lebo kvôli istote potrebujeme $k + 1$ pokusov pre vstup (x, y) taký, že $Num(x) - Num(y) = p_1 \cdot p_2 \cdot \dots \cdot p_k$

Pre použitie metódy svedkov sú teda vhodné také typy úloh, ktoré spĺňajú nasledujúce podmienky

- existencia svedkov
- efektívna konštrukcia kandidáta+test, či je alebo nie je svedkom
- množina kandidátov bohatá na svedkov

*Freivaldsova
metóda*

Metóda odtláčkov je vlastne špeciálnym prípadom metódy svedkov. Používa sa, keď nás zaujíma rovnosť/ekvivalencia nejakých komplexných objektov.

1. Vezmeme množinu M vhodných zobrazení úplných reprezentácií do čiastočných reprezentácií (napr. $N \rightarrow Prim(n^2)$)
 $h \leftarrow random(M)$
2. vypočítame redukované reprezentácie $h(O_1)$, $h(O_2)$; tomu sa hovorí odtláčok/stopa/fingerprint
3. porovnáme skrátené reprezentácie
if $h(O_1) = h(O_2)$ **then** return("ekvivalentné")
else return ("nie sú ekvivalentné")

K úspešnému použitiu potrebujeme

- $h(O)$ efektívne počítateľné
- porovnanie $h(O_1) \stackrel{?}{=} h(O_2)$ efektívne vypočítateľné
- existenciu kandidátov/svedkov v M , ktoré potvrdzujú, že $O_1 \neq O_2$

Neprekvapí, že máme tradeoff medzi dĺžkou $h(O)$ a pravdepodobnosťou korektnej odpovede.

náhodná vzorka

Náhodná vzorka/Random sampling využívame v situácii, keď nevieme hľadať objekty daných vlastností ale vieme, že ich je veľa. Potom náhodne vybraná množina kandidátov by s veľkou pravdepodobnosťou mala hľadaný objekt obsahovať.

Táto pravdepodobnostná metóda je založená na dvoch jednoduchých faktoch:

Fakt 4.1 *Ku každej náhodnej premennej X existuje hodnota, ktorá nie je menšia (väčšia) ako $E[X]$*

Fakt 4.2 Ak pre náhodný objekt je pravdepodobnosť, že má nejakú vlastnosť, nenulová, potom existuje objekt, ktorý túto vlastnosť má.

Zvyšovanie úspešnosti opakovaním je realizované znižovaním pravdepodobnosti chyby nezávislým opakovaním behov, resp. len ich častí, na tom istom *opakovaní* vstupe

Optimizácia a náhodné zaokrúhľovanie Niekedy je optimalizačný problém ťaž- *zaokrúhľovanie* ký na množine diskretných vstupov, ale jeho riešenie na \mathbb{R} je efektívne (napr. lineárne programovanie). V tejto situácii problém často riešime tak, že relaxujeme od diskretných hodnôt, vyriešime problém v reálnych číslach a získané riešenie vhodne zaokrúhlime. (V aprox alg—primal-dual metóda pre ..)

Kapitolou samou o sebe je derandomizácia—prehľadný a efektívny pravdepodobnostný algoritmus "derandomizujeme" na deterministický; väčšinou simulovaním všetkých možných behov pravdepodobnostného algoritmu.

4.1 Eliminácia protihráča

V tejto časti sa budeme venovať metóde eliminácie protihráča. Sústredíme sa na konštrukciu triedy deterministických algoritmov pre ktoré platí, že *pre každý vstup je väčšina z nich efektívna*. Náhodná voľba jedného z nich je potom s veľkou pravdepodobnosťou pre konkrétny vstup efektívna. Aplikujeme na dva príklady.

Pre problém hašovanie ukážeme, že

hašovanie

- ku každej hašovacej funkcii existuje zlý vstup
- existuje trieda hašovacích funkcií H taká, že pre každý vstup S a náhodnú hašovaciu funkciu $h \in H$, je $h(S)$ "dobro" rozložené

Pre problém plánovania ukážeme, že pravdepodobnostný algoritmus dáva kvalita- *on line plánova-* tívne lepšie riešenie ako ľubovoľný deterministický algoritmus. *nie*

4.1.1 Hašovanie

Uvažujme problém data managementu, keď podľa kľúča k hľadáme v štruktúre T . Operácie, ktoré potrebujeme realizovať, sú—Search(T,k), Insert(T,k), Delete(T,k). Ide o implementáciu tzv. slovníka, keď na implementáciu množiny objektov použijeme tabuľku spolu s vhodným hašovaním.

- tabuľka T s priamym prístupom, $T = \{0, 1, \dots, m - 1\}$
- univerzum $U = \{0, 1, \dots, d\}$; $|U| \ggg |T|$
- zaujíma nás taká hašovacia funkcia/zobrazenie $h : U \rightarrow T$, pre ktorú je $S \subseteq U, |S| = n$, a pritom vyžadujeme, aby S bola "dobro rozložená v T ": v priemere $\frac{|S|}{|T|} = \frac{n}{m}$ prvkov zahašovaných do jednej bunky/hodnoty
- nech b -ta bunka má zoznam ℓ prvkov zahašovaných na hodnotu b ; potom pre zložitosť realizácie jednotlivých inštrukcií platí — najhorší prípad ℓ , expected $\ell/2$

Čo vyžadujeme od zobrazenia h ?

- dá sa efektívne vypočítať
- pre väčšinu $S \subseteq U$ prvky "rovnomerne rozhadzuje"; inými slovami

$$T(i) = \{a \in S \mid h(a) = i\} \Rightarrow |T_i| = O\left(\frac{|S|}{|T|}\right)$$

Uvedomme si, že nič lepšie ako "pre väčšinu S " žiadať nemôžeme, nakoľko pre $S = U_{h,i} = \{a \in U \mid h(a) = i\}$ sa celá množina S zobrazí do jednej bunky...

Lema 4.3 *Nech $U = \mathbb{N}$, $T = \{0, 1, \dots, m-1\}$, $n \in \mathbb{N}^+$, $h : U \rightarrow T$ spĺňa: $Pr[h(x) = i] = \frac{1}{m}$. Potom*

- očakávaný počet prvkov v jednej bunke je $\frac{n}{m} + 1$
- ak $n = m$, potom $Pr[\text{viac ako jeden prvok } h(x) = i] < 1/2$

Dôkaz: Vezmime $S = \{s_1, \dots, s_n\}$ náhodne z U

$$X_{i,j}^l(S) = \begin{cases} 1, & \text{ak } h(s_i) = h(s_j) = l \\ 0, & \text{inak} \end{cases}$$

$$E[X_{i,j}^l] = Pr[h(s_i) = l]Pr[h(s_j) = l] = \frac{1}{m^2}$$

Ohraničíme počet kolízií v l -tej bunke

$$E[X^l] = \sum_{1 \leq i < j \leq n} E[X_{i,j}^l] = \sum_{1 \leq i < j \leq n} \frac{1}{m^2} = \frac{n(n-1)}{2m^2} < \frac{n^2}{2m^2}$$

$$n = m \Rightarrow E[X^l] < 1/2$$

Nech k prvkov z S ide do l -tej bunky; potom máme $\binom{k}{2}$ kolízií

$$\frac{n^2}{2m^2} > E[X^l] = \frac{k(k-1)}{2} > \frac{(k-1)^2}{2} \quad \text{z čoho úpravou } \frac{n}{m} + 1 > k$$

□

Ak je teda $n \sim m$, tak očakávaná zložitost' je $O(1)$. Reálne data ale nie sú veľmi rovnomerne rozložené...

K jednej hašovacej funkcii sa zlý vstup konštruuje ľahko. Budeme preto používať *rozumnú* množinu hašovacích funkcií a náhodný výber; to konštrukciu zlého vstupu skomplikuje...

univerzálna trieda hašovacích funkcií **Definícia 4.1** *Nech H je konečná množina hašovacích funkcií z U do $T = \{0, 1, \dots, m-1\}$. Hovoríme, že H je univerzálna trieda funkcií, ak $\forall x, y \in U, x \neq y$, platí*

$$|\{h \in H \mid h(x) = h(y)\}| \leq \frac{|H|}{m}$$

O tom, že univerzálna trieda hašovacích funkcií je definovaná rozumne, hovorí nasledujúca veta.

Veta 4.4 *Nech $S \subseteq U$, $|S| = n$, $T = \{0, 1, \dots, m-1\}$, H je univerzálna trieda hašovacích funkcií. Potom pre každé $x \in S$ a náhodnú hašovaciu funkciu $h \in H$ pre očakávanú veľkosť množiny $S_x(h) = \{a \in S \mid a \neq x, h(a) = h(x)\}$ platí*

$$E[|S_x(h)|] \leq \frac{|S|}{|T|} = \frac{n}{m}$$

Dôkaz: Uvažujme náhodnú premennú $Z_{x,y}(h)$, ktorá pre hašovaciu funkciu h identifikuje kolíziu medzi x, y a premennú $Z_x(h)$, ktorá spočíta počet kolízií s prvkom x . Stredná hodnota $Z_x(h)$ určuje očakávaný počet kolízií.

$$Z_{x,y}(h) = \begin{cases} 1, & h(x) = h(y) \\ 0, & h(x) \neq h(y) \end{cases} \quad E[Z_{x,y}(h)] = Pr[h(x) = h(y)] \leq 1/m$$

$$Z_x(h) = \sum_{y \in S, x \neq y} Z_{x,y}(h) \quad E[Z_x(h)] = \sum_{y \in S, x \neq y} E[Z_{x,y}(h)] \leq \frac{|S| - 1}{m} < \frac{n}{m}$$

□

Univerzálna trieda hašovacích funkcií existuje, stačí zobrať všetky funkcie $U \rightarrow T$.

Lema 4.5 *Trieda $H_{U,T} = \{h \mid h : U \rightarrow T\}$ je univerzálna trieda hašovacích funkcií.*

Dôkaz: Nech $|T| = m$

- $|H_{U,T}| = m^{|U|}$
- $|H(x, y, i)| = |\{h \mid h(x) = h(y) = i\}| = m^{|U|-2}$
- $|H(x, y)| = \{h \mid h(x) = h(y)\} = \sum_{i=0}^{m-1} |H(x, y, i)| = m^{|U|-1} = \frac{|H_{U,T}|}{m}$

□

Hoci je trieda $H_{U,T}$ univerzálna, pre naše účely nie je vhodná. Dôvod?

- je príliš veľká
- väčšina funkcií nemá efektívnu reprezentáciu

Existujú aj univerzálne triedy hašovacích funkcií, ktoré sú praktické. Zoberme ako univerzum $U = \{0, 1, \dots, p-1\}$, hašovacia tabuľka T je veľkosti m , pričom p, m sú H_{lin}^p prvočísla. Ukážeme, že vhodnou univerzálnou triedou hašovacích funkcií je napr. nižšie definovaná trieda hašovacích funkcií H_{lin}^p :

$$h_{a,b}(x) = ((ax + b) \bmod p) \bmod m$$

$$H_{lin}^p = \{h_{a,b} \mid a \in \{1, 2, \dots, p-1\}; b \in \{0, 1, \dots, p-1\}\}$$

Veta 4.6 *Pre každé prvočíslo p (a číslo m) je trieda H_{lin}^p univerzálnou triedou hašovacích funkcií z $U = \{0, 1, \dots, p-1\}$ do $T = \{0, 1, \dots, m-1\}$*

Dôkaz: Označme

$$H_{lin}^p(x, y) = \{h_{a,b} \in H_{lin}^p, h_{a,b}(x) = h_{a,b}(y)\}$$

$$M(x, y) = \{(r, s) \in U \times U \mid r \neq s \text{ a } r \equiv s \pmod{m}\}$$

Zrejme $|H_{lin}^p(x, y)| = |M(x, y)|$. Potrebujeme ukázať, že $\forall x, y : |H_{lin}^p(x, y)| \leq \frac{|H_{lin}^p|}{m}$. Uvažujme najprv zjednodušenú verziu $h'_{a,b}(x) = (ax + b) \bmod p$.

1. Ukážeme, že $f_{x,y}(a, b) = (h'_{a,b}(x), h'_{a,b}(y))$ je bijekcia; $a, b \overset{h'_{a,b}}{\longleftrightarrow} r, s$
2. Spočítame, koľko je k danému r takých s , že $r \neq s \bmod p$ ale $r = s \bmod m$

1. Podme argumentovať, že $f_{x,y}$ je bijekcia. Nech $r = h'_{a,b}(x) = (ax+b) \bmod p$, $s = h'_{a,b}(y) = (ay+b) \bmod p$.

Kvôli sporu predpokladajme, že $x \neq y$ a $r = s$. Potom

$$x \neq y \Rightarrow r \neq s$$

$$0 = r - s \equiv a(x - y) \bmod p$$

Keďže p je prvočíslo, znamenalo by to $a \equiv 0 \bmod p$ alebo $x \equiv y \bmod p$, čo vzhľadom k voľbe a, x, y nie je možné. Preto $x \neq y \Rightarrow r \neq s$. To ale znamená, že $f_{x,y}$ je zobrazenie z $\underbrace{U \times \{U - \{0\}\}}_{ls} \rightarrow \underbrace{\{(r, s) | r, s \in U, r \neq s\}}_{rs}$. Keďže $|ls| = p(p-1) = |rs|$,

stačí ukázať, že jedna z $f_{x,y}, f_{x,y}^{-1}$ je injektívna.

$$r \neq s \Rightarrow a \neq b$$

Nech $r \neq s$. Keďže p je prvočíslo, pre $x \neq y, 0 \leq x, y \leq p-1$ inverzný prvok k $(x-y)$ existuje :

$$\begin{aligned} r - s = a(x - y) \bmod p &\Rightarrow a = (r - s)(x - y)^{-1} \bmod p \\ &\Rightarrow b = (r - ax) \bmod p \end{aligned}$$

2. Pre x, y voľba náhodne zvolenej $h_{a,b}$ jednoznačne určuje dvojicu (r, s) takú, že $(r, s) = (h'_{a,b}(x), h'_{a,b}(y))$, preto $|H_{lin}^p(x, y)| = |M(x, y)|$. K výpočtu $|H_{lin}^p(x, y)|$ preto stačí spočítať, koľko je takých $r \neq s$, že $r \equiv s \bmod m$.

K jednému r je $\lceil \frac{p}{m} \rceil \leq \frac{p+m-1}{m} = \frac{p-1}{m} + 1$ takých s , že $r \equiv s \bmod m$. Preto

$$|H_{lin}^p(x, y)| = |M(x, y)| \leq \frac{p(p-1)}{m} = \frac{|H_{lin}^p|}{m}$$

□

Vec

Ďalším príkladom je trieda hašovacích funkcií, ktorú konštruujeme k univerzu $U(r, m)$, kde m je prvočíslo, $r \in \mathbb{N}$

$$U(m, r) = \{x = (x_0, \dots, x_r); 0 \leq x_i \leq m-1, i = 0, \dots, r\} = T^{r+1}$$

Nech $\alpha = (\alpha_0, \dots, \alpha_r) \in T^{r+1}$; definujeme

$$\text{Vec} = \{h_\alpha; \alpha \in T^{r+1}\}, \text{ kde } h_\alpha(x_0, \dots, x_r) = \left(\sum_{i=0}^r \alpha_i x_i \right) \bmod m$$

Lema 4.7 *Vec je univerzálna množina hašovacích funkcií*

Dôkaz: Ukážeme, že $\forall x \neq y$ je $|H_\alpha(x, y)| = |\{h_\alpha \in \text{Vec} \mid h_\alpha(x) = h_\alpha(y)\}| \leq \frac{|\text{Vec}|}{m} = m^r$. Ak $x \neq y$, tak sa tieto vektory líšia aspoň v jednej zložke; pre jednoduchosť nech $x_0 \neq y_0$.

$$\begin{aligned} h_\alpha(x) = h_\alpha(y) &\Leftrightarrow \sum_{i=0}^r \alpha_i x_i \equiv \sum_{i=0}^r \alpha_i y_i \bmod m \\ \alpha_0(x_0 - y_0) &\equiv \sum_{i=1}^r \alpha_i (y_i - x_i) \bmod m \end{aligned}$$

Keď m je prvočíslo, existuje pre $x_0 \neq y_0$ inverzný prvok $(x_0 - y_0)^{-1}$. Preto

$$\alpha_0 \equiv \sum_{i=1}^r \alpha_i (y_i - x_i) (x_0 - y_0)^{-1} \bmod m$$

Hodnota α_0 je teda jednoznačne určená hodnotami $x, y, \alpha_1, \dots, \alpha_r$, čo znamená, že $|H_\alpha(x, y)| \leq m^r$. □

Positívne na množine Vec je to, že

- vieme efektívne generovať náhodné h_α
- h_α sa efektívne počíta.

4.1.2 On line algoritmy

On line problém je taký, keď postupnosť vstupov prichádza postupne a rozhodnutia treba robiť priebežne. Otázka je, nakoľko dobrý môže byť algoritmus, ktorý nepozná dopredu budúcnosť v porovnaní s tým, ktorý celú budúcnosť pozná dopredu. Čo považujeme za kvalitu?

kvalita on-line algoritmov

- kvalita riešenia
- zložitosť

dispečing obmedzený počet sanitiek/taxíkov/... obsluhuje požiadavky naň. Dispečer priebežne rozhoduje, koho kam pošle. Kritéria optimalizácie

motivačné príklady

- celkový počet najazdených km
- čas čakania pacientov/pasažierov -celkový resp. maximalizovaný na jedného

rozvrhovanie/scheduling analogicky - máme niekoľko strojov rônej kvality a požiadavky na ne. Pridelujeme úlohy strojom, pričom optimalizujeme

- celkový čas
- priemerné zaťaženie
- čas čakania

Majme optimalizačný problém $U = (I, Sol, m, cieľ)$. Algoritmus A je online pre U, ak pre každý vstup $x = x_1, \dots, x_n \in I$

on-line algoritmus

- x_1, \dots, x_i je prípustný vstup $\forall 1 \leq i \leq n$
 - $A(x) \in Sol(x)$
 - $A(x_1, \dots, x_i)$ je časť A(x) $\forall 1 \leq i \leq n$
- vstup $x_1, \dots, x_n \rightsquigarrow$ výstup $y_1, \dots, y_n, y_i = f_i(x_1, \dots, x_i)$

Kvalitu algoritmu A meriame vzhľadom k optimálnemu off-line algoritmu, resp. jeho cene. $Comp_A(x)$

$$Comp_A(x) = \max \left\{ \frac{OPT_U(x)}{m_A(x)}, \frac{m_A(x)}{OPT_U(x)} \right\}$$

V prípade aproximačných algoritmov definujeme chybu a aproximačný prah ako hranicu "najlepšej" kvality, v prípade on-line algoritmov máme analogické pojmy.

Nech $\delta \geq 1$ je konštanta. Hovoríme, že on-line algoritmus A je δ -competitive, ak $\forall x Comp_A(x) \leq \delta$. On-line problém je δ -ťažký, ak neexistuje d -competitive algoritmus na jeho riešenie pre žiadne $d < \delta$.

δ -ťažký problém

Príklad 4.1 Majme cash/buffer veľkosti k , ostatné stránky nech sú v pomalej pamäti. Zo vstupu prichádzajú požiadavky na stránky. Ak požadovaná stránka s nie je v buffri, musíme niektorú zo stránok v buffri vyhodiť a presunúť tam požadovanú stránku s ; v takomto prípade máme presun. Chceme minimalizovať počet presunov medzi cash a hlavnou pamäťou.

stránkovanie

začiatok: s_1, \dots, s_k v Cash
 s_{k+1}, \dots, s_n v Main; $n \gg k$

inštancia: postupnosť indexov i_1, i_2, \dots, i_m ; $1 \leq i_j \leq n$

cieľ: minimalizovať počet presunov medzi Cash a Main

Ukážeme, že problém Stránkovanie je k -ťažký. Spravíme to popisom stratégie protihráča, ktorý k ľubovoľnému algoritmu A skonštruuje "zlý" vstup x_A

protihráč

1. pri požiadavke $(k + 1)$ na stránku s_{k+1} musí algoritmus A vyhodiť niektorú stránku z buffra-povedzme stránku s indexom j_1

2. po vyhodení stránky s_{j_1} prichádza od protihráča požiadavka j_1 ; algoritmus A vyhadzuje j_2
- ⋮

Je zrejmé, že nech by sme mali akýkoľvek algoritmus, takto konštruovaný vstup $k + 1, j_1, \dots, j_{k-1}$ vyžaduje k presunov. Pritom optimálne—offline—riešenie v prvom kroku vyhodí $i \in \{1, \dots, k\} - \{j_1, \dots, j_{k-1}\}$, preto mu na spracovanie rovnakej postupnosti stačí len jeden presun.

$$\text{Comp}_A(x_A) = \frac{\text{cost}A(x_A)}{\text{OPT}(x_A)} = \frac{k}{1} = k$$

◇

Definícia 4.2 Pravdepodobnostný algoritmus A je pravdepodobnostný online algoritmus pre problém U ak $\forall x_1 \dots x_n \in L$ a $\forall 1 \leq i \leq n - 1$

- výstup každého behu $A(x_1, \dots, x_i)$ je prípustné riešenie
- pre každý vstup $C(x_1, \dots, x_i), x_{i+1}$, kde $C(x_1, \dots, x_i) \in \text{Sol}(x_1, \dots, x_i)$, všetky behy A dopočítajú prípustné riešenie pre x_1, \dots, x_{i+1}

$$\begin{aligned} \text{vstup } x_1 \dots x_i x_{i+1} &\rightsquigarrow \text{výstup } y_1, \dots, y_{i+1} \\ &y_{i+1} = f_{i+1}(x_1, \dots, x_{i+1}) = g_{i+1}(f_i(x_1, \dots, x_i), x_{i+1}) \end{aligned}$$

Pre inštanciu vstupu X máme

$\mathbf{S}_{A,X}$ je množina výpočtov A na X

$\mathbf{Prob}_{A,X}$ je odpovedajúce pravdepodobnostné rozdelenie na $S_{A,X}$

$\mathbf{Z}_X(\mathbf{C}) = \text{Comp}_C(X)$ je náhodná premenná v $(S_{A,X}, \text{Prob}_{A,X})$

Kvalitu meriame očakávanou hodnotou

$\mathbf{Exp} - \mathbf{Comp}_A(\mathbf{X}) = E[Z_x]$

Nech $\delta \in R$. Hovoríme, že algoritmus A je $\mathbf{E}[\delta]$ -**competitive**, ak

$$\text{Exp} - \text{Comp}_A(X) = E[Z_X] \leq \delta$$

Nech $h : \mathbb{N} \rightarrow R^{\geq 1}$. Hovoríme, že algoritmus A je $\mathbf{Exp}[h]$ -**competitive**, ak

$$\text{Exp} - \text{Comp}_A(X) = E[Z_X] \leq h(|x|)$$

Cieľom tejto časti je na probléme rozvrhovania ukázať, že existujú problémy, pre ktoré pravdepodobnostný algoritmus dosahuje lepšiu kvalitu ako najlepší možný deterministický algoritmus.

Problém Rozvrhovanie predpokladá

problém

- máme m rôznych (typov) strojov M_1, \dots, M_m , pričom z každého typu stroja je len jeden kus
- *job* je m úloh A_1, \dots, A_m reprezentovaných permutáciou indexov i_1, \dots, i_m ; *job* (i_1, \dots, i_m) znamená, že
 - úlohy treba počítat v poradí A_1, \dots, A_m ; najskôr jednu úlohu dokončiť, potom druhú začať
 - A_j sa musí riešiť na stroji M_{i_j}
 - idealizovane predpokladáme, že výpočet na každom stroji trvá presne jednotku času

Unit(m,d), resp. skrátene $U(m, d)$

vstup: m typov strojov, d jobov(m-tíc úloh)

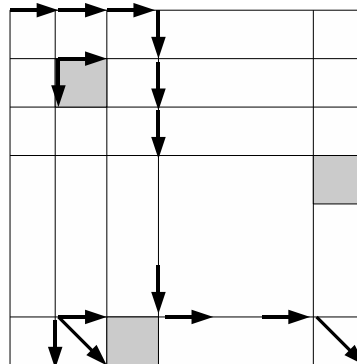
výstup: pridelenie úloh jednotlivým strojom v diskretnom čase tak, aby každý robil vždy len na jednej úlohe a aby príslušná úloha mala k dispozícii všetky potrebné výsledky (zachovanie poradia)

Budeme uvažovať najjednoduchšiu verziu úlohy $U(m, 2)$. Vstupom sú dva vektory $\alpha = (i_1, \dots, i_m)$ a $\beta = (j_1, \dots, j_m)$. Predstavme si ich ako $m \times m$ mriežku $G(\alpha, \beta)[j, k]$, pričom riadky odpovedajú β , stĺpce α . **Kolízia** nastáva, ak $G[k, \ell] = j_k = i_\ell$

Ľahko vidno, že ak úlohy j_1, \dots, j_{k-1} a $i_1, \dots, i_{\ell-1}$ skončili naraz a $j_k \neq i_\ell$, potom možno úlohy j_k a i_ℓ počítať paralelne; inak jedna z nich musí počkať.

Do mriežky na miesta kolízií dáme značku. Hýbať sa možno smerom doprava a dole a to po riadkoch, stĺpcoch a diagonálach, ak tam nie je značka. Priradenie je cesta z ľavého horného do pravého spodného rohu.

Pohyb po horizontálnej šípke aj pohyb po vertikálnej šípke odpovedajú tomu, že jedna úloha stojí. Keďže máme štvorcovú mriežku, je počet vertikálnych a horizontálnych šípok na každej ceste rovnaký.



Majme teda cestu S

$$\left. \begin{array}{l} del_\alpha(S), \text{ počet vertikálnych hrán na ceste } S \\ del_\beta(S), \text{ počet horizontálnych hrán na ceste } S \end{array} \right\} delay(S) = del_\alpha(S) = del_\beta(S)$$

$$cost(S) = m + delay(S) = m + \frac{del_\alpha(S) + del_\beta(S)}{2}$$

Fakt 4.8 Pre $m \in \mathbb{N}$ má každá inštancia (α, β) pre $U(m, 2)$ presne m konfliktov, pričom v každom riadku a každom stĺpci je práve jeden konflikt.

Najprv ukážeme, že vieme konštruovať zlé vstupy.

Lema 4.9 Nech $m = 0 \pmod 8$. Pre každý online algoritmus A riešiaci $U(m, 2)$ existuje taká inštancia $I = ((1, 2, \dots, m), \beta)$, že

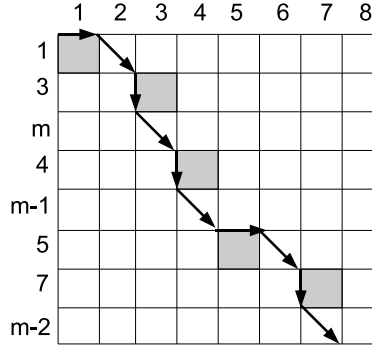
$$cost(A(I)) \geq m + \frac{m}{8}$$

Dôkaz: Popíšeme stratégiu protivníka, ktorý bude ukladať konflikty tak, že časť cesty, ktorá prejde za $m/2$ ty riadok a $m/2$ -ty stĺpec, použije aspoň polovicu hrán, ktoré nie sú diagonálne. To bude znamenať, že nabrala zdržanie aspoň polovicu z $\frac{1}{2} \frac{m}{2}$, čo je požadovaných $m/8$.

Konštruujeme $\beta = j_1, \dots, j_m$

- $j_1 = 1$ spôsobí, že máme konflikt. Algoritmus môže značku obísť po vertikálnej alebo horizontálnej hrane, oboje spôsobí prírastok 1 do príslušnej $del_\alpha(S)$ alebo $del_\beta(S)$
- Ak algoritmus obišiel značku po horizontálnej hrane, α má riešiť úlohu 2, β úlohu 1–toto sa rieši paralelne. Sme v pozícii, keď α ide riešiť úlohu 3 a my chceme, aby aj β riešila úlohu 3–dáme teda $j_2 = 3$

- Ak algoritmus obišiel značku po vertikálnej hrane, α má riešiť úlohu 1, β novú úlohu j_2 . Aby sa to dalo riešiť paralelne, položíme $j_2 = m$ a potom voľba $j_3 = 2$ spôsobí opäť konflikt.



Takto postupujeme ďalej. Ak algoritmus obchádza prekážku po horizontálnej hrane, doplníme na príslušné miesto v β prvok z množiny $\{1, 2, \dots, m/2\}$. Ak ju obíde po vertikálnej hrane, budeme vkladat prvok z množiny $\{m/2 + 1, \dots, m\}$ \square

Teraz odhadneme zložitosť najlepšieho offline deterministického algoritmu. Vezme množinu konkrétnych deterministických stratégií, spočítame strednú hodnotu pri ich použití. Potom optimálny algoritmus nemôže byť horší ako táto stredná hodnota.

Lema 4.10 *Nech $m \in \mathbb{N}$. Pre každý vstup I do $U(m, 2)$ platí*

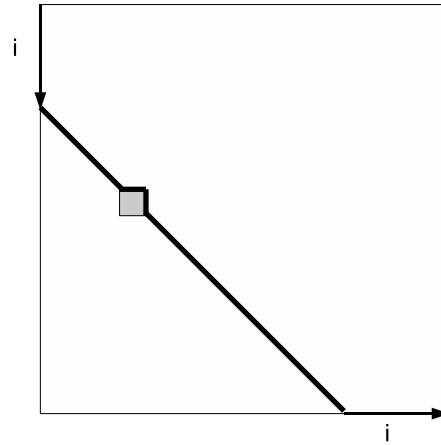
$$Opt(I) \leq m + \sqrt{m}$$

Dôkaz: Kvôli zjednodušeniu uvažujme $m = k^2$.

Pre $i = 0, 1, \dots, k$ označme D_i diagonálu, ktorá ide z $(0, i)$ do $((m-i, m))$
 D_{-i} diagonálu, ktorá ide z $(i, 0)$ do $((m, m-i))$

Stratégia $A(j), j \in \{-\sqrt{m}, \dots, 0, \dots, \sqrt{m}\}$ príde po obvodě k diagonále D_i , potom sa snaží ísť po diagonále D_i a následne po obvodě najkratšou cestou do pravého spodného rohu. Ak sú na diagonále prekážky, obchádza ich jednou horizontálnou a jednou vertikálnou hranou.

Analogicky stratégia $A(-i)$ (pozri obr.)



Pre cenu stratégie $A(i)$ potom platí $cost(A(i)) = |i| + (m - |i|) + |i| + \underbrace{\text{počet prekážok}}_{\text{zdržanie}} = m + |i| + \text{počet prekážok}$

Keďže je počet všetkých prekážok m , pre súčet diagonálnych stratégií dostávame

$$m + \sum_{i=-\sqrt{m}}^{\sqrt{m}} |i| = m + 2 \sum_{i=1}^{\sqrt{m}} |i| = m + 2 \frac{\sqrt{m}(\sqrt{m} + 1)}{2} = 2m + \sqrt{m}$$

Priemerný počet zdržaní je potom

$$\frac{2m + \sqrt{m}}{2\sqrt{m} + 1} < \frac{2m + \sqrt{m}}{2\sqrt{m}} = \sqrt{m} + 1/2$$

Existuje teda algoritmus so zdržaním nanajvyš \sqrt{m} a optimálny od neho nemôže byť horší. Preto $Opt(I) \leq m + \sqrt{m}$. \square

Na základe lemy 4.9 a lemy 4.10 máme nasledujúce tvrdenie.

Veta 4.11 *Problém $U(m, 2)$ je $(9/8 - \varepsilon)$ -ťažký.*

Dôkaz: Z lemy 4.9 vieme, že ku každému algoritmu A existuje inštancia I , na ktorej je $cost_A(I) \geq m + \frac{m}{8}$. Lema 4.10 zas ohraničuje cenu optimálneho algoritmu. Preto

$$Comp_A(I) = \frac{cost_A(I)}{Opt(I)} \geq \frac{\frac{9}{8}m}{m + \sqrt{m}} = \frac{9}{8} \left(1 - \underbrace{\frac{1}{\sqrt{m} + 1}}_{\varepsilon} \right)$$

\square

A teraz konečne ten pravdepodobnostný algoritmus

Algoritmus 11 DIAG

- 1: náhodne zvol $i \in \{-\sqrt{m}, \dots, \sqrt{m}\}$
 - 2: rieš stratégiou $A(i)$
-

Veta 4.12 *Algoritmus DIAG je pravdepodobnostný online, pričom pre každú inštanciu vstupu I do $U(m, 2)$ platí*

$$ExpComp_{DIAG}(I) \leq 1 + \frac{1}{\sqrt{m}} + \frac{1}{2m}$$

Dôkaz: Na základe predchádzajúcich úvah

- očakávaný počet zdržaní pre stratégiu $A(i)$ je nanajvyš $\lceil \sqrt{m} \rceil + 1/2$.
- očakávaný čas je nanajvyš $m + \lceil \sqrt{m} \rceil + 1/2$
- $Opt \geq m$

$$ExpComp_{DIAG}(i) = \frac{\text{očakávaná cena}}{Opt(I)} \leq \frac{m + \lceil \sqrt{m} \rceil + 1/2}{m} = 1 + \frac{1}{\sqrt{m}} + \frac{1}{2m}$$

\square

O probléme $U(m, 2)$ sme ukázali

- je $(9/8 - \varepsilon)$ -ťažký
- existuje pravdepodobnostný algoritmus s ExpComp nanajvyš $1 + \frac{1}{\sqrt{m}} + \frac{1}{2m}$

záverom

4.2 Metóda odtlačkov

Ako sme už povedali, metóda odtlačkov sa používa, keď nás zaujíma rovnosť/ekvivalencia nejakých komplexných objektov.

1. Vezmeme množinu M vhodných zobrazení úplných reprezentácií do čiastočných reprezentácií (napr. $N \rightarrow Prim(n^2)$)
 $h \leftarrow random(M)$
2. vypočítame redukované reprezentácie $h(O_1)$, $h(O_2)$; tomu sa hovorí odtlačok/stopa/fingerprint
3. porovnáme skrátene reprezentácie
if $h(O_1) = h(O_2)$ **then return** ("ekvivalentné")
else return ("nie sú ekvivalentné")

K úspešnému použitiu potrebujeme

- $h(O)$ efektívne počítateľné
- porovnanie $h(O_1) \stackrel{?}{=} h(O_2)$ efektívne vypočítateľné
- existenciu dostatočne veľkej množiny kandidátov/svedkov v M , ktorí v prípade $O_1 \neq O_2$ potvrdzujú, že $O_1 \neq O_2$

Dôraz je na konštrukcii tej množiny M . Jej kardinalita je určená dĺžkou popisu jednotlivých objektov a tak proti sebe idú snaha o dostatočné množstvo kandidátov a snaha o krátky popis.

4.2.1 Porovnávanie databáz

Aplikáciou metódy odtlačkov je Algoritmus 2 na porovnanie dvoch vzdialených databáz; odtlačkom je mod p pre zvolené prvočíslo p . Analogickým spôsobom môžeme riešiť viaceré podobné príklady

Príklad 4.2 *Majme vzdialené databázy. X, U_i, V_j označujú podľa kontextu reťazec ale aj číslo s binárnym zápisom X , resp. U_i, V_j*

R_I : $X = x_1 \dots x_n$

R_{II} : $U = \{U_1, \dots, U_k\}$; $U_i \in \{0, 1\}^n$

Zaujíma nás, či $X \in U$.

Množina zobrazení/vytváraní odtlačkov vzniká využitím prvočísel – pre prvočíslo p označuje h_p funkciu, ktorá ako odtlačok berie mod p

$$M = \{h_p \mid p \in Prim(n^2)\}$$

Algoritmus je jasný

chyba

Je zřejmé, že keď $X \in U$, ľubovoľná voľba h_p (výpočet $C(p)$) vedie ku korektnej odpovedi. Chyby sa môžeme dopustiť len vtedy, ak $X \notin U$ ale existuje U_i tak, že $X \equiv U_i \pmod p$. Vieme, že pre konkrétne i je počet takýchto "zlých" prvočísel nanejvýš $n - 1$, pričom n je dĺžka zápisu X . Ak teda označíme

$$A_i = \{p \mid p \in Prim(n^2) \ \& \ s = q_i v \text{ behu } C(p)\}$$

$$A = \bigcup_{i=1}^k A_i$$

Algoritmus 12 — Test $X \in U$

RI:

- 1: náhodne vyber $h_p \in M$
- 2: $s \leftarrow X \pmod p$
- 3: pošli s, p druhému počítaču RII

RII:

- 1: vypočítaj $q_i \leftarrow U_i \pmod p$
- 2: **if** $s \in \{q_1, \dots, q_k\}$ **then** return " $X \in U$ "
- 3: **else** return " $X \notin U$ "

tak

$$\text{Error}(X, U) = \text{Prob}(A) = \sum_{i=1}^k \text{Prob}(A_i) \leq \sum_{i=1}^k \frac{2 \ln n}{n} = k \cdot \frac{2 \ln n}{n}$$

Pre $k \leq \frac{n}{4 \ln n}$ je pravdepodobnosť chyby nanajvyšš $1/2$. Inými slovami, ak mohutnosť množiny U je nanajvyšš $\frac{n}{4 \ln n}$, je Algoritmus 4.2 Monte Carlo algoritmus s jednosmernou chybou.

Analogicky riešime problém neprázdneho prieniku dvoch databáz.

Príklad 4.3 Majme vzdialené databázy

RI: $V = \{V_1 \dots V_\ell\}; V_i \in \{0, 1\}^n$

RII: $U = \{U_1, \dots, U_k\}; U_i \in \{0, 1\}^n$

Zaujímá nás $V \cap U \stackrel{?}{=} \emptyset$.

Algoritmus 13 — Test $V \cap U$

RI:

- 1: náhodne vyber $h_p \in M$
- 2: $s_i \leftarrow V_i \pmod p$
- 3: pošli s_1, \dots, s_ℓ, p druhému počítaču RII

RII:

- 1: vypočítaj $q_i \leftarrow U_i \pmod p$
- 2: **if** $\{s_1, \dots, s_\ell\} \cap \{q_1, \dots, q_k\} = \emptyset$ **then** return " $V \cap U = \emptyset$ "
- 3: **else** return " $V \cap U \neq \emptyset$ "

Algoritmus prekomunikuje $(\ell + 1)2 \lceil \log n \rceil$ bitov.

komunikácia

Je zrejme, že keď $V \cap U = \emptyset$, ľubovoľná voľba h_p (výpočet $C(p)$) vedie ku korektnej chyba odpovedi. Chyby sa môžeme dopustiť len vtedy, ak $V \cap U \neq \emptyset$, ale existujú U_i, V_j tak, že $U_i \equiv V_j \pmod p$. Ak teda označíme B_i udalosť, keď $s_i \in \{q_1, \dots, q_k\}$, tak

$$\text{Error}(V, U) = \text{Prob}\left(\bigcup_{i=1}^{\ell} B_i\right) \leq \sum_{i=1}^{\ell} k \cdot \frac{2 \ln n}{n} = \ell k \cdot \frac{2 \ln n}{n}$$

Pre $\ell k = o\left(\frac{n}{\ln n}\right)$ máme Monte Carlo algoritmus s jednosmernou chybou.

Uvedomme si, že ak zväčšíme množinu M tak, že budeme uvažovať prvočísla $p \in \text{Prim}(n^d)$, počet "zlých" prvočísel sa nezmení, ale zmenší sa pravdepodobnosť chyby. Pre $X = x_1 \dots x_n, Y = y_1 \dots y_n$ totiž

$$\text{Pr}[X \equiv Y \pmod p \mid X \neq Y] \leq \frac{n-1}{\text{Prim}(n^d)} \leq \frac{d \ln n}{n^{d-1}}$$

Vo všetkých uvedených príkladoch sme objekty porovnávali na základe "odtlačkov", ktorými boli zvyšky po delení prvočíslom. Iným prístupom je Freivaldsova metóda, ktorá ako odtlačky používa hodnotu polynómu/matice/funkcie v bodoch.

4.2.2 Freivaldsova metóda - ekvivalencia polynómov

Základom Freivaldsovej metódy je porovnanie objektov na základe odtlačku, ktorým je hodnota polynómu/matice/funkcie ... Rozumnú pravdepodobnosť chyby potom dosahujem počítaním nad vhodným okruhom...

Príkladom využitia Freivaldsovej metódy bol Algoritmus 1 na porovnanie $AB = C$ pre matice A, B, C . Zvolili sme náhodný vektor α a odpoveď na otázku $AB \stackrel{?}{=} C$ sme spravili na základe výsledku porovnania $AB\alpha \stackrel{?}{=} C\alpha$. Analogický postup môžeme využiť pre porovnávanie polynómov (a iných štruktúr, ktoré sa na porovnanie polynómov dajú transformovať).

Vstupom sú polynómy $P_1(X), P_2(X)$ stupňa najvyššie n . Zaujímá nás, či $P_1(X) = P_2(X)$. Ak chceme deterministický algoritmus na porovnanie polynómov, využijeme ich "normálny tvar"

$$P(X) = \sum_{i=0}^n c_i X^i$$

resp.

$$Q(x_1, \dots, x_n) = \sum_{i_1=0}^d \dots \sum_{i_n=0}^d c_{i_1} \dots c_{i_n} x_1^{i_1} \dots x_n^{i_n}$$

pre polynóm $Q(x_1, \dots, x_n)$ n premenných, z ktorých každá môže byť stupňa najvyššie d . Toto deterministické porovnanie má tú nevýhodu, že "normálny" tvar môže byť až exponenciálne dlhý vzhľadom k zadanému tvaru polynómu; napr. $(x_1 + x_2)(x_1 + x_3) \dots (x_1 + x_n)$

Uvažujme teda jednoduchý algoritmus, ktorý porovná polynómy na základe ich hodnoty v náhodnom bode, prípadne náhodných bodoch.

Algoritmus 14 — Test $P_1(X) = P_2(X)$

Nech polynómy P_1, P_2 stupňa n majú premenné z poľa F , nech $S \subseteq F$, $|S| \geq n + 1$

- 1: náhodne vyber $r \in S$
 - 2: $p_1 \leftarrow P_1(r)$, $p_2 \leftarrow P_2(r)$
 - 3: **if** $p_1 = p_2$ **then** return " $P_1(X) = P_2(X)$ "
 - 4: **else** return " $P_1(X) \neq P_2(X)$ "
-

chyba

Potrebuje odhadnúť pravdepodobnosť chybnjej odpovede. Ak sú polynómy rovnaké, chyby sa evidentne nedopustíme. Chyba nastane vtedy, ak pre polynóm $Q(X) = P_1(X) - P_2(X)$, ktorý nie je identicky rovný 0, zvolíme r tak, že $Q(r) = 0$.

Fakt 4.13 *Polynóm stupňa d má najvyššie d rôznych koreňov.*

Na základe faktu 4.13 je pravdepodobnosť chyby Algoritmu 14 najvyššie $\frac{n}{|S|}$. Ak chceme chybu zmenšiť, môžeme zväčšiť množinu S , resp. opakovať niekoľko behov algoritmu. Uvedomme si, že ak $Q(X) \neq 0$, potom $n + 1$ behov s rôznymi hodnotami r to musí potvrdiť.

"ponaučenie"

Zhrnutím dostávame, že test na rovnosť polynómov $P_1(X) \stackrel{?}{=} P_2(X)$ je rozumné transformovať na test $P_1(X) - P_2(X) \stackrel{?}{=} 0$

Pri prechode k polynómom viacerých premenných využijeme nasledujúcu vetu.

Veta 4.14 (Schwartz-Zippel) *Nech $Q(x_1, \dots, x_n) \in F[x_1, \dots, x_n]$ je polynóm viacerých premenných celkového stupňa d . Nech $S \subseteq F$ a r_1, \dots, r_n sú náhodne vybrané z S . Potom*

$$\Pr[Q(r_1, \dots, r_n) = 0 \mid Q(x_1, \dots, x_n) \neq 0] \leq \frac{d}{|S|}$$

Dôkaz: Postupujeme indukciou vzhľadom na počet premenných n .

Polynóm stupňa d má najviac d rôznych koreňov. ✓

$n=1$

Nech $k \leq d$ je maximálny stupeň exponentu premennej, povedzme že je to x_1 . $n > 1$

Potom

$$Q(x_1, \dots, x_n) = \sum_{i=0}^k x_1^i Q_i(x_2, \dots, x_n)$$

Vzhľadom na výber vieme, že

- koeficient $Q_k(x_2, \dots, x_n)$ pri x_1^k nie je nula
- stupeň polynómu $Q_k(x_2, \dots, x_n)$ je najviac $d - k$
- pravdepodobnosť, že $Q_k(r_2, \dots, r_n) = 0$ je najviac $\frac{d-k}{|S|}$

Nech teda $Q_k(r_2, \dots, r_n) \neq 0$. Uvažujme polynóm

$$q(x_1) = Q(x_1, r_2, \dots, r_n) = \sum_{i=0}^k x_1^i Q_i(r_2, \dots, r_n)$$

Stupeň polynómu $q(x_1)$ je najviac k , nie je identicky rovný 0, preto

$$\Pr[q(r_1) = 0 \mid q(x_1) \neq 0] \leq \frac{k}{|S|}$$

Využijúc $\Pr[A] \leq \Pr[A|\bar{B}] + \Pr[B]$ dostávame

$$\Pr[Q(r_1, \dots, r_n) = 0 \mid Q(x_1, \dots, x_n) \neq 0] \leq \frac{d-k}{|S|} + \frac{k}{|S|} = \frac{d}{|S|}$$

□

Ak počítame mod p , kde p je prvočíslo, chybu môžeme odhadnúť pomocou nasledujúcej lemy.

Lema 4.15 *Nech p je prvočíslo, $Q(x_1, \dots, x_n) \neq 0$ polynóm nad Z_p , $d \in \mathbb{N}$ je maximálny stupeň premených. Potom Q má najviac $n \cdot d \cdot p^{n-1}$ koreňov.*

Dôkaz: Analogicky ako v predchádzajúcom dôkaze-indukcia cez počet premenných.

Počet koreňov je najviac $d = ndp^{n-1}$

$n=1$

$Q(x_1, \dots, x_n) = \sum_{i=0}^d x_1^i Q_i(x_2, \dots, x_n)$. Ak $Q(\alpha_1, \dots, \alpha_n) \equiv 0 \pmod{p}$, tak alebo $n > 1$

- (a) $Q_i(\alpha_2, \dots, \alpha_n) \equiv 0 \pmod{p}$ pre každé i , alebo
- (b) $Q_j(\alpha_2, \dots, \alpha_n) \not\equiv 0 \pmod{p}$, pričom α_1 je koreňom polynómu

$$q(x_1) = Q_0(\alpha_2, \dots, \alpha_n) + x_1 Q_1(\alpha_2, \dots, \alpha_n) + \dots + x_1^d Q_d(\alpha_2, \dots, \alpha_n)$$

Spočítame obe možnosti zvlášť. Keďže $Q(x_1, \dots, x_n) \not\equiv 0$, musí existovať taký index k , že polynóm $Q_k(x_2, \dots, x_n) \not\equiv 0$. Ten má podľa IP najviac $(n-1)dp^{n-2}$ (a) koreňov takých, že $Q_i(\alpha_2, \dots, \alpha_n) \equiv 0 \pmod{p}$. Keď dovolíme α_1 ľubovoľne, je

počet koreňov $(\alpha_1, \dots, \alpha_n)$ v prípade (a) nanajvyš $(n-1)dp^{n-1}$.

Polynóm $q(x_1)$ je polynóm jednej premennej stupňa d , preto má nanajvyš d koreňov. V tomto prípade máme teda nanajvyš dp^{n-1} koreňov. (b)

Celkovo teda je koreňov nanajvyš

$$(n-1)dp^{n-1} + dp^{n-1} = ndp^{n-1}$$

□

Algoritmus 14 ľahko upravíme na Monte Carlo algoritmus s jednosmernou chybou, ktorý rieši porovnanie polynómov viacerých premenných.

Algoritmus 15 AQP

Polynómy P_1, P_2 n premenných nad Z_p , maximálny stupeň jednotlivých premenných d , prvočíslo p

- 1: náhodne vyber $\alpha \in \{0, 1\}^n$
 - 2: $h_\alpha(P_1) \leftarrow P_1(\alpha) \pmod p$
 - 3: $h_\alpha(P_2) \leftarrow P_2(\alpha) \pmod p$
 - 4: **if** $h_\alpha(P_1) = h_\alpha(P_2)$ **then** return " $P_1 \equiv P_2$ "
 - 5: **else** return " $P_1 \not\equiv P_2$ "
-

Pravdepodobnosť, že v prípade $Q() = P_1() - P_2() \not\equiv 0$ zvolíme náhodne α tak, že $Q(\alpha) \not\equiv 0$, je aspoň $\left(1 - \frac{nd}{p}\right)$.

Uvedený algoritmus môžeme využiť na pravdepodobnostné zodpovedanie otázky, či má daný bipartitný graf $G(U, V, E)$, $E \subseteq U \times V$ úplné párovanie (perfect matching)¹.

úplné párovanie

Úplné párovanie je zrejme možné len vtedy, keď $|U| = |V|$. Predpokladáme teda, že $U = \{u_1, \dots, u_n\}$, $V = \{v_1, \dots, v_n\}$. Úplné párovanie je vtedy dané permutáciou π , ktorá hovorí, že v párovaní sú hrany $(u_i, v_{\pi(i)})$. Využijeme Edmondsonovu vetu

Veta 4.16 (Edmondson) *Nech A je matica typu $n \times n$, ktorá k bipartitnému grafu $G(U, V, E)$ skonštruujeme nasledovne*

$$A[i, j] = \begin{cases} x_{ij}, & (u_i, v_j) \in E \\ 0, & (u_i, v_j) \notin E \end{cases}$$

Definujme polynóm² $Q(x_{11}, \dots, x_{nn}) = \det(A)$. Potom G má úplné párovanie práve vtedy ak $Q \neq 0$.

Dôkaz: $\det(A) = \sum_{\pi \in S(n)} -1^{\text{sgn}(\pi)} A[1, \pi(1)] A[2, \pi(2)] \dots A[n, \pi(n)]$; tu $S(n)$ je symetrická grupa permutácií a $\text{sgn}(\pi)$ je parita počtu výmen, ktorými z identickej permutácie dostaneme permutáciu π . Keďže každé x_{ij} je v matici nanajvyš raz, sumáciou sa nemôžu "vynulovať". Preto determinant nie je identicky rovný 0 práve vtedy, keď existuje permutácia, pre ktorú je príslušný člen nenulový. To je ale ekvivalentné tomu, že

¹Párovanie je taká množina hrán, v ktorej je každý vrchol obsiahnutý nanajvyš raz. Párovanie je úplné, ak je v ňom každý vrchol obsiahnutý práve raz.

² $\det(A)$ označuje determinant matice A .

$$\left. \begin{array}{l} A[1, \pi(1)] \neq 0 \Rightarrow (1, \pi(1)) \in E \\ A[2, \pi(2)] \neq 0 \Rightarrow (2, \pi(2)) \in E \\ \dots \\ A[n, \pi(n)] \neq 0 \Rightarrow (n, \pi(n)) \in E \end{array} \right\} \text{úplné párovanie}$$

□

Determinant je polynóm, preto testovanie determinantu je testom polynómu na nulu a na to máme pravdepodobnostný test. Využitie tohto prístupu—súvis párovania a determinantu matice/polynómu—je skôr v konštrukcii paralelného algoritmu na hľadanie úplného párovania.

4.2.3 Perfect matching

V tejto časti sa budeme venovať hľadaniu nezávislej množiny hrán:

- maximálne párovanie – nedá sa zväčšiť pridaním hrany; sekvenčne sa ľahko hľadá greedy metódou
- kardinalitou maximálne párovanie; existujú polynomiálne sekvenčné riešenia. Zdá sa, že všetky rozumné paralelizácie obsahujú pravdepodobnostný prístup.
- perfect matching– obsahuje všetky vrcholy grafu

Budeme sa venovať problému úplného párovania v grafe, o ktorom vieme, že toto párovanie obsahuje. Môžeme si to dovoliť, pretože rozhodovacia verzia tohto problému je z RNC.

Definícia 4.3 *Trieda RNC je trieda jazykov, pre ktoré existuje PRAM A taký, že RNC, ZNC $\forall x \in \Sigma^*$*

- $x \in L \Rightarrow Pr[A(x) \text{ je akceptuj}] \geq 1/2$
- $x \notin L \Rightarrow Pr[A(x) \text{ je akceptuj}] = 0$
- *plynomiálny počet procesorov*
- *polylogaritmický čas*

Pravdepodobnostný algoritmus je založený na nasledujúcej vete:

Veta 4.17 (Tutte) *Nech A je matica, ktorá vznikla z grafu $G = (V, E)$:*

- *každý hrane $(v_i, v_j) \in E$ priradíme premennú $x_{i,j}, i < j$*
- *hodnota matice na mieste i, j je určená podľa hrán*

$$A[i, j] = \begin{cases} x_{i,j}, & (v_i, v_j) \in E, i < j \\ -x_{i,j}, & (v_i, v_j) \in E, j < i \\ 0, & (v_i, v_j) \notin E, i < j \end{cases}$$

Potom G obsahuje úplné párovanie práve vtedy, keď determinant matice A, $\det(A)$ nie je identicky rovný 0.

Keďže existuje NC algoritmus na výpočet determinantu matice, poskytuje veta 5.9 návod na RNC riešenie rozhodovacej verzie existencie úplného párovania v grafe:

- konštrukcia matice A
- pravdepodobnostný test na $\det(A) \neq 0$

Zopakujeme(pripomeňme) si niektoré pojmy a tvrdenia súvisiace s maticami

definície, fakty

(i,j) minor $U^{i,j}$ je matica, ktorá z matice U vznikne odstránením i -teho riadku a j -teho stĺpca

adjoint $adj(U)$ je matica, ktorá má na mieste (i, j) hodnotu v absolútnej hodnote rovnú determinantu (i, j) -minora;

$$adj(U)(i, j) = (-1)^{i+j} \det(U^{i,j})$$

$$\text{Platí } U \cdot adj(U) = \det(U)$$

Veta 4.18 *Nech U je $n \times n$ matica, ktorej prvky sú k -bitové čísla. Potom determinant, adjoint, U^{-1} sú z NC. Presnejšie, ak $M(n) = O(n^{2.376})$ je počet operácií spotrebovaných pri násobení 2 matíc typu $n \times n$, potom*

– determinant počítame v čase $O(\log^2 n)$ pomocou $O(n^2 M(n))$ procesorov

– existuje RNC pre výpočet inverznej matice, adjoint v čase $O(\log^2 n)$ pomocou $O(n^{3.5}k)$ procesorov.

Problémom pri získavaní paralelného algoritmu na hľadanie úplného párovania v grafe, o ktorom vieme, že úplné párovanie má, je to, že týchto úplných párovaní môže byť veľa. Potrebujeme nejakým spôsobom "izolovať" jedno, ktoré bude viacero procesorov hľadať spoločne. Túto izoláciu spravíme vhodným ohodnotením hrán a následne hľadaním úplného párovania minimálnej váhy. Ukážeme, že pri rozumne náhodnom váhovaní je veľká šanca, že sa izolácia podarí.

system podmnožín

Definícia 4.4 *Systém podmnožín je dvojica (X, \mathcal{F}) , kde $X = \{x_1, \dots, x_m\}$ je konečná množina prvkov, tzv. univerzum a $\mathcal{F} = \{S_1, \dots, S_k, S_i \subseteq X\}$ je trieda podmnožín. Hovoríme, že rozmer tohto systému je m -veľkosť univerza.*

Nech $w : X \rightarrow \mathbb{Z}^+$ je váhová funkcia. Definujme

$$w(S) = \sum_{x_j \in S} w(x_j)$$

Platí nasledujúca izolačná lema:

Lema 4.19 (izolačná) *Nech (X, \mathcal{F}) je systém podmnožín rozmeru m , $w : X \rightarrow \{1, \dots, 2m\}$ je kladná celočíselná váhovacia funkcia, ktorá každému prvku z X priradí náhodnú váhu z množiny $\{1, \dots, 2m\}$. Potom*

$$\Pr[\exists \text{ jediná množina minimálnej váhy v } \mathcal{F}] \geq 1/2$$

Dôkaz: Predpokladáme, že každý prvok z X sa vyskytuje aspoň v jednej z podmnožín systému \mathcal{F} . Predstavme si, že váhy jednotlivým prvkom priraďujeme postupne. Nech všetky prvky z X až na x_i už majú priradenú váhu. Označme

W_i minimálnu váhu množiny, ktorá obsahuje x_i , pričom váha $w(x_i) = 0$, keďže ju nepoznáme

\overline{W}_i minimálnu váhu množiny, ktorá neobsahuje x_i

$$\alpha(i) = \overline{W}_i - W_i \quad (\text{všimnime si, že } \alpha(i) \text{ môže byť aj } < 0)$$

Ako sa prejaví priradenie váhy prvku x_i ?

- Ak by sme priradili x_i nekonečne malú váhu (resp. $-2m^2$), potom by každá množina minimálnej váhy musela obsahovať prvok x_i . Analogicky, nekonečne veľká váha (resp. $2m^2$) by spôsobila, že žiadna minimálna množina x_i neobsahuje.
- $w(x_i) < \alpha(i)$ spôsobí, že každá množina minimálnej váhy musí obsahovať x_i , pričom každá množina, ktorá x_i neobsahuje, má váhu aspoň $W_i + w(x_i) < W_i + (\overline{W}_i - W_i) = \overline{W}_i$

- $w(x_i) > \alpha(i)$ spôsobí, že žiadna množina minimálnej váhy neobsahuje x_i , pričom každá množina, ktorá x_i obsahuje, má váhu aspoň $W_i + w(x_i) > \overline{W}_i$

Ak $w(x_i) = \alpha(i)$, nevieme jednoznačne povedať, či x_i bude alebo nebude prvkom minimálnej množiny. V tomto prípade hovoríme, že je x_i *neisté*.

Ak však $w(x_i) < \alpha(i)$, potom je prítomnosť/nepřítomnosť x_i v minimálnej množine jednoznačne určená.

$$w(x_i) = \alpha(i)$$

$$w(x_i) \neq \alpha(i)$$

Uvedomme si, že náhodná premenná $\alpha(i)$ je nezávislá od $w(x_i)$, preto

$$Pr[x_i \text{ je neisté}] \leq 1/2m$$

Neistoty jednotlivých prvkov korelujú, preto

$$Pr[\text{existuje neistý prvok v } X] \leq m \cdot \frac{1}{2m} = \frac{1}{2}$$

Znamená to, že popísané náhodné priradenie váh je s pravdepodobnosťou aspoň $1/2$ také, že žiaden prvok nie je neistý, čo znamená, že existuje jediná množina minimálnej váhy. (Existencia dvoch množín S_i, S_j minimálnej váhy by znamenala, že existuje prvok $y \in S_i, y \notin S_j$, čo by znamenalo, že y je neistý.)

□

Aplikácia izolačnej lemy na problém úplného párovania je priamočiara

- X je množina hrán
- \mathcal{F} je množina úplných párovanií
- náhodné priradenie váh jednotlivým hranám určí jednoznačne úplné párovanie minimálnej váhy.

Môžeme teda predpokladať, že máme graf s náhodne priradenými váhami, ktorý obsahuje jediné úplné párovanie minimálnej váhy (keďže nejednoznačné je to s pravdepodobnosťou $\leq 1/2$, niekoľkonásobné opakovanie chybu redukuje; resp. máme Las Vegas algoritmus, ktorý to ováňovanie nakoniec predsalen nájde.)

paralelizácia

Nech A je Tutte matica. Vytvoríme z nej maticu B tak, že namiesto $x_{i,j}$ píšeme $2^{w(i,j)}$. Potom platia nasledujúce tvrdenia:

Lema 4.20 *Nech existuje jediné minimálne úplné párovanie váhy W . Potom $\det(B) \neq 0$ a súčasne maximálna mocnina dvojky, ktorá delí $\det(B)$, je 2^{2W} .*

Lema 4.21 *Nech M je jediné minimálne úplné párovanie v G váhy W . Potom*

$$(i, j) \in M \Leftrightarrow \frac{\det(B^{i,j})2^{w(i,j)}}{2^{2W}} \text{ je nepárne}$$

Zhrnutím predchádzajúcich úvah a tvrdení je MC algoritmus P-Match, pre ktorý platí:

Veta 4.22 *Ak G má úplné párovanie, potom Algoritmu P-Match ho nájde s pravdepodobnosťou aspoň $1/2$. Přitom čas je $O(\log^2 n)$ a počet procesorov $O(n^{3.5}m)$*

4.2.4 Porovnávanie reťazcov

Venujme sa teraz problému porovnávaní/vyhľadávania textov (pattern matching).

vstup: $X = x_1 \dots x_n \quad x_i \in \Sigma$

$Y = y_1 \dots y_m \quad y_j \in \Sigma$

výstup k pozícia výskytu Y v X : $x_k \dots x_{k+m-1} = Y$

$k = n - m + 2$ znamená, že Y sa v X nevyskytuje

Algoritmus 16 P-Match

▷ vstupom je graf s aspoň jedným úplným párovaním

-
- 1: **for all** (i, j) (paralelne) **do**
 - 2: priradiť hrane (i, j) náhodnú váhu $w(i, j) \in \{1, \dots, 2m\}$
 - 3: vypočítaj (deterministicky) Tutte maticu B
 - 4: vypočítaj $\det(B)$
 - 5: vypočítaj W také, že 2^{2W} je maximálna mocnina dvojky, ktorá delí $\det(B)$
 - 6: vypočítaj $\text{adj}(B) = \det(B) \cdot B^{-1}$
 - 7: **for all** (i, j) (paralelne) **do**
 - 8: vypočítaj $r_{i,j} = \det(B^{i,j}) \cdot 2^{w(i,j)} / 2^{2W}$
 - 9: **if** (i, j) je nepárne **then** pridaj $r_{i,j}$ do M
-

"Klasické" deterministické prístupy sú dva

1. $O(n \cdot m)$ algoritmus "hrubej sily" založený na prikladaní Y postupne na pozície 1, 2 až $n - m + 1$
2. $O(n + m)$ algoritmus využívajúci predspracovanie Y

vzorka pomocou
odtlačkov

Metódu odtlačkov vieme využiť v (asymptoticky) neefektívnejšej verzii deterministického algoritmu – vzorku Y budeme postupne prikladať na jednotlivé pozície v X , porovnanie príslušných podreťazcov však spravíme cez ich odtlačky získané funkciou O_p . Opäť využijeme počítanie modulo prvočíslo p .

$$\begin{array}{|c|} \hline \mathbf{x}_k \dots \mathbf{x}_{k+m-1} \\ \hline Y \\ \hline \end{array}$$

$$\mathbf{x}_k \dots \mathbf{x}_{k+m-1} ? Y \iff O_p(\mathbf{x}_k \dots \mathbf{x}_{k+m-1}) ? O_p(Y)$$

Pre jednoduchosť predpokladáme, že $\Sigma = \{0, 1\}$. Potom reťazec dĺžky m môžeme vnímať ako binárny zápis čísla a jeho odtlačkom bude zvyšok po delení prvočísлом p . Nech $X(j)$ označuje číslo, ktorého binárny zápis je podreťazec dĺžky m reťazca X , ktorý začína na pozícii j ; analogicky Y .

Vezmime prvočíslo $p \leq \tau$ náhodne, $O_p(X) = X \bmod p$. Ak by sme rovnosť odtlačkov považovali za definitívne určenie pozície, na ktorej sa Y v X nachádza, chyba algoritmu by bola

$$\sum_{j=1}^{n-m+1} Pr[O_p(Y) = O_p(X(j))]$$

Keďže pre binárne číslo dĺžky m máme nanajvyš $m - 1$ "zlých" prvočísel, môžeme písať

$$\sum_{j=1}^{n-m+1} Pr[O_p(Y) = O_p(X(j)) \mid Y \neq X(j)] < \frac{nm}{\text{Prim}(\tau)} = O\left(\frac{nm \log \tau}{\tau}\right)$$

To pri voľbe $\tau = f(n, m) = n^2 m \log n^2 m$ dáva pravdepodobnosť chyby MC algoritmu s jednosmernou chybou

$$\frac{nm \log(n^2 m \log n^2 m)}{n^2 m \log n^2 m} \leq \frac{\log(n^2 m)^2}{\log n^2 m n} = \frac{2 \log n^2 m}{n \log n^2 m} = \frac{2}{n}$$

resp. $O(1/n)$.

Ak však rovnosť odtlačkov použijeme len ako indikáciu *potenciálneho* výskytu Y v X a skutočnosť overíme "priložením" Y na príslušnú pozíciu, dostaneme algoritmus

Algoritmus 17 String(f(n,m))

▷ f určuje veľkosť použitých prvočísel

```

1: náhodne vyber  $p \in Prim(f(n, m))$ 
2:  $O_p(Y) \leftarrow Y \pmod p$ 
3:  $O_p(X(0)) \leftarrow Num(x_1 \dots x_{m-1}) \pmod p$ 
4:  $k \leftarrow 1$ 
5: while  $k < n - m + 1$  do
6:    $O_p(X(k)) \leftarrow (2[O_p(X(k-1)) - x_{k-1}2^{m-1} \pmod p] + x_{k+m-1}) \pmod p$ 
    $\triangleright O_p(X) \leftarrow X(k) \pmod p; x_0 = 0$ 
7:   if  $O_p(Y) = O_p(X(k))$  then
8:     if  $Y = X(k)$  then return "Y sa vyskytuje od pozície k"
      $\triangleright$  z pravdepodobnostného algoritmu robíme deterministický
9:   else  $k \leftarrow k + 1$ 
10: return "Y sa v X nevyskytuje"

```

Las Vegas, ktorého každá odpoveď je korektná. V tomto prípade má zmysel sa pýtať, aká je jeho očakávaná časová zložitosť.

Pri realizácii algoritmu využívame efektívne počítanie $X(j) \pmod p$, ktoré vychádza z rovnosti

$$X(j+1) = 2[X(j) - 2^{m-1}x_j] + x_{j+m}$$

Analyzujeme Las Vegas verziu (Algoritmus 17). Očakávaná zložitosť je

čas

$$O \left(\underbrace{(n+m) \left(1 - \frac{1}{n}\right)}_{\text{žiaden potenciálny výskyt}} + \underbrace{mn \left(\frac{1}{n}\right)}_{\text{overovanie na každej pozícii}} \right) = O(n+m)$$

Upravme algoritmus tak, že po falošnej zhode vygenerujeme prvočíslo p náhodne. Potom pravdepodobnosť toho, že spravíme viac ako t reštartov, je nanajvyš $\frac{1}{n^t}$.

modifikácia Las Vegas

4.2.5 Interaktívne dôkazy

Poslednou (našou) aplikáciou metódy odtlačkov sú tzv. *interaktívne dôkazy*. Predpokladajme, že niekto tvrdí, že pozná dôkaz. Našou úlohou je sa presvečiť, že ho pozná bez toho, aby nám ho celý ukázal. Môžeme vidieť iba "odtlačok" tohto dôkazu, pričom odtlačkom sú odpovede na otázky, ktoré budeme klásť. Kvalita dôkazov, ktoré takýmto spôsobom dokážeme overiť, zrejme závisí od kvantity odpovedí a kvality otázok. Ukážeme jeden "vzorový" príklad—problém izomorfizmu a neizomorfizmu grafov.³

Problém izomorfizmu grafov (GI): dvojicu vstupných grafov (G_1, G_2) akceptujeme, ak sú izomorfné. Ľahko vidno, že tento problém patrí do NP: uhádneme izomorfizmus τ a overíme rovnosť $\tau(G_1) = G_2$.

Problém neizomorfizmu grafov (GNI) dvojicu vstupných grafov (G_1, G_2) akceptujeme, ak nie sú izomorfné. Tento problém je z co-NP. Nemáme krátke dôkazy, asi je to ťažšie...

Na dokazovanie použijeme systém dvoch hráčov/algoritmov.

³Grafy $G_1 = (V, E_1)$, $G_2 = (V, E_2)$ sú izomorfné, ak existuje permutácia π taká, že $(i, j) \in E_1 \Leftrightarrow (\pi(i), \pi(j)) \in E_2$

V-verifier je pravdepodobnostný polynomiálny algoritmus, ktorý sa snaží overiť, že grafy G_1, G_2 nie sú izomorfné. Môže pritom klásť otázky dôkazu P

P-proover je algoritmus, ktorému neohraničujeme výpočtovú silu. Zakážeme mu jediné a to prístup k náhodným bitom algoritmu V

výpočet je komunikácia medzi nimi, pričom posledné slovo má V. Vyžadujeme

- ak G_1, G_2 sú neizomorfné, potom P má šancu presvedčiť V
- ak G_1, G_2 sú izomorfné, potom akákoľvek snaha P vedie k akceptovaniu s pravdepodobnosťou nanajvyš $1/2$

Takémuto algoritmu hovoríme interaktívny dôkaz. Komunikačný protokol (IP) pre V a P pre problém GNI je jednoduchý.

V pracuje takto

- uhádne $i \in \{1, 2\}$ a permutáciu τ
- vypočíta $H = \tau(G_i)$
- pošle H do P a spýta sa na index i

P pošle V index j

V porovná si indexy i, j . Ak $i = j$, akceptuje, že G_1, G_2 sú neizomorfné; inak zamietá

Veta 4.23 *Ak G_1, G_2 nie sú izomorfné, čestný P presvedčí V. Inak je pravdepodobnosť toho, že (hoci aj nečestný) P presvedčí V, nanajvyš $1/2$.*

Dôkaz: Analyzujeme podľa reálnej situácie

$G_1 \approx G_2$

Ak grafy nie sú izomorfné, výpočtovo neohraničene silný P dokáže nájsť aj korektný index i aj permutáciu τ , ktorou vzniklo $H = \tau(G_i)$. V tomto prípade V odpovedá korektne.

$G_1 \sim G_2$

Ak sú grafy izomorfné, potom $G_1 \sim H \sim G_2$. Akokoľvek silný P bez prístupu k náhodným bitom V nedokáže zistiť index, ktorý po ňom V chce. P si preto tipne, čo vedie k pravdepodobnosti chyby nanajvyš $1/2$. \square

Len pre zaujímavosť. Označme IP vyššie popísaný systém (V, P). Dá sa (nie celkom jednoducho) ukázať, že $IP = PSPACE$.

4.3 Zvyšovanie úspešnosti opakovaním a náhodná vzorka

Tieto dve metódy – zvyšovanie úspešnosti opakovaním a náhodný vzorka – sú natoľko podobné, že niekedy ťažko rozlíšiť, o ktorú z nich ide. Navyše, ich vhodná kombinácia dáva dobré výsledky.

Doteraz sme nezávislé opakovanie celých výpočtov využívali k znižovaniu chyby pod želanú hranicu. Metódu teraz potiahneme ďalej *opakovanie*

- opakovanie nie celých výpočtov, ale len vybraných častí
- opakovanie rôznych častí výpočtu rôzny počet krát: častejšie zopakujeme tie časti, v ktorých je pravdepodobnosť výskytu chyby vyššia

Táto metóda poskytuje "rozumné" riešenie pre viaceré problémy, pre ktoré rozum- *náhodná vzorka*
né/efektívne deterministické algoritmy nepoznáme

4.3.1 Zlepšovanie úspešnosti opakovaním kritických častí

Teraz sa budeme venovať problému Min-Cut (hranový rez minimálnej ceny). Zopakujme si definíciu problému:

vstup: multigraf $G = (V, E, c)$ $c : E \rightarrow \mathbb{N} - \{0\}$ určuje násobnosť hrán *problém Min-Cut*

prípustné riešenia: $M(G) = \{(V_1, V_2) \mid V_1 \cup V_2 = V, V_1 \cap V_2 = \emptyset\}$
je to množina hranových rezov

cena: $cost((V_1, V_2), G) = \sum_{\ell \in S(V_1, V_2)} c(\ell)$
 $S(V_1, V_2) = \{(x, y) \in E \mid x \in V_1, y \in V_2\}$

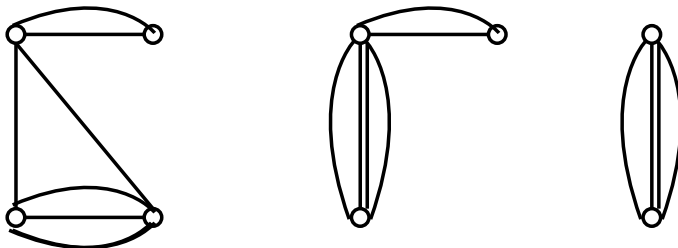
cieľ: minimalizácia ceny

Najlepší známy deterministický algoritmus je zložitosti $O(|V| \cdot |E| \cdot \log \frac{|V|^2}{|E|})$, čo je v najhoršom prípade $O(n^3)$. Pravdepodobnostný algoritmus je vlastne náhodným generovaním hranového rezu, o ktorom ukážeme, že je s veľkou pravdepodobnosťou minimálnym. Samotný algoritmus je založený na *kontrakcii hrán*. Nech (x, y) je hrana; jej kontrakciou rozumieme

$contract(G, (x, y))$:

- zlúčenie vrcholov x, y do jedného vrchola $x \cup y$
- odstránenie vrcholov x, y a hrán (x, y)
- presmerovanie hrán idúcich do/z x alebo y do/z vrchola $x \cup y$

Takto vzniknutý graf označíme $G/(x, y)$. Nech $F = \{(x_1, y_1), \dots, (x_k, y_k)\} \subseteq E^+$. Graf, ktorý vznikne postupným kontrahovaním hrán $(x_1, y_1), \dots, (x_k, y_k)$ označíme G/F . Všimnime si, že na poradí kontrakcie hrán nezáleží.



Algoritmus na výpočet minimálneho rezu je jednoduchý - náhodne zvolíme hrana, správime kontrakciu a opakujeme, kým graf neobsahuje dva vrcholy. Ako výsledok vezmeme množinu hrán medzi týmito vrcholmi (Algoritmus 18)

Algoritmus 18 Contraction

```

1: label(v) ← v
2: while G má viac ako 2 vrcholy do
3:   náhodne vyber hranu e = (x, y) ∈ E(G)
4:   G ← contract(G, e)
5:   label(z) ← label(x) ∪ label(y), kde z je kontrakciou novovzniknutý vrchol
6: nech G = ({u, v}, E(G)): return((label(u), label(v)), |E(G)|)

```

Veta 4.24 Algoritmus 18 je polytime pravdepodobnostný algoritmus, ktorý optimálne rieši problém Min-Cut s pravdepodobnosťou aspoň $\frac{2}{n(n-1)}$, kde n je počet vrcholov vstupného grafu G.

Dôkaz: Začneme analýzou časovej zložitosti

čas

- jedna kontrakcia je úmerná prezretiu hrán prislúchajúcich k dvom vrcholom; keďže násobné hrany reprezentujeme pomocou funkcie $c \implies O(n)$
- počet opakovaní je daný počtom vrcholov pôvodného grafu - skončíme, keď kontrahovaný graf má 2 vrcholy $\implies n - 2$

$O(n^2)$

korektnosť

Analýzu úspešnosti/chyby spravíme spočítaním pravdepodobnosti, že hrany z fixovaného minimálneho rezu C_{min} ostanú neskontrahované. Nech $cost(C_{min}) = k$. Pri argumentácii využijeme nasledujúce –zjavné– fakty

- Ak k je cena minimálneho hranového rezu, tak G má aspoň $\frac{nk}{2}$ hrán (každý vrchol má stupeň aspoň k, hrana má dva konce)
- Algoritmus 18 počíta $C_{min} \iff$ žiadna hrana z $E(C_{min})$ nebola kontrahovaná

Označme

$$E_i = \{\text{výpočty, ktoré v } i\text{-tej iterácii nevybrali na kontrakciu hranu z } E(C_{min})\}$$

Zaujímá nás pravdepodobnosť udalosti $\bigcap_{i=1}^{n-2} E_i$

$$Pr \left[\bigcap_{i=1}^{n-2} E_i \right] = Pr [E_1] \cdot Pr [E_2 \mid E_1] \cdot Pr [E_3 \mid E_1 \cap E_2] \cdots Pr \left[E_{n-2} \mid \bigcap_{i=1}^{n-3} E_i \right]$$

Postupne:

$$Pr [E_1] = \frac{|E| - |E(C_{min})|}{|E|} = 1 - \frac{k}{|E|} \geq 1 - \frac{k}{\frac{kn}{2}} = 1 - \frac{2}{n}$$

Po $i - 1$ iteráciách—náhodných kontrakciách—máme $(n - i + 1)$ vrcholov, pričom každý z nich stále musí mať stupeň aspoň k. Ostal nám graf G/F_{i-1} , v ktorom je aspoň $\frac{k(n-i+1)}{2}$ hrán. Z nich vyberáme jednu, pričom $|E(C_{min})|$ z nich je "zlých"

$$Pr \left[E_i \mid \bigcap_{j=1}^{i-1} E_j \right] \geq \frac{|E(G/F_{i-1}) - |E(C_{min})||}{|E(G/F_{i-1})|} \geq 1 - \frac{k}{|E(G/F_{i-1})|} \geq 1 - \frac{k}{\frac{k(n-i+1)}{2}} = 1 - \frac{2}{n-i+1}$$

$$\begin{aligned} Pr \left[\bigcap_{j=1}^{n-2} E_j \right] &\geq \prod_{i=1}^{n-2} \left(1 - \frac{2}{n-i+1} \right) = \prod_{\ell=n}^3 \frac{\ell-2}{\ell} = \\ &= \frac{n-2}{n} \cdot \frac{n-3}{n-1} \cdot \frac{n-4}{n-2} \cdots \frac{3}{5} \cdot \frac{2}{4} \cdot \frac{1}{3} = \frac{2}{n(n-1)} = \frac{1}{\binom{n}{2}} \end{aligned}$$

□

Máme teda v $O(n^2)$ garantovanú úspešnosť $\frac{2}{n(n-1)} > \frac{2}{n^2}$. To znamená, že ak zopakujeme $n^2/2$ nezávislých behov tohto algoritmu a ako výsledok vezmeme najlepšie z riešení, chyba bude s pravdepodobnosťou nanajvyšš

$$\left(1 - \frac{2}{n^2}\right)^{n^2/2} < \frac{1}{e}$$

Lenže čas je $O(n^4)$

Uvedený prístup je "naivný" v tom, že sme opakovali celé výpočty. Zamyslime sa nad tým, dokedy má zmysel robiť kontrakciu a kedy je už graf dostatočne malý na to, aby sme minimálny rez – pre tento kontrahovaný graf – mohli deterministicky dopočítať. Uvažujme funkciu $\ell(n)$, ktorá určuje čas ukončenia kontrakcií/počet realizovaných kontrakcií – počet vrcholov grafu, pri ktorom zvyšok dopočítame deterministicky. Vyžadujeme, aby $1 < \ell(n) < n$, ℓ monotónna parameter $\ell(n)$

Algoritmus 19 DetRan(ℓ)

- 1: aplikuj Algoritmus 18 na G dovtedy, kým nemá $\ell(n)$ vrcholov; výsledkom je graf G/F
 - 2: aplikuj na G/F s $\ell(n)$ vrcholmi najlepší deterministický algoritmus D
 - 3: return (D(G/F))
-

Analyzujeme časovú zložitosť Algoritmu 19.

$$\left. \begin{array}{l} O((n - \ell(n)) \cdot n), \text{ príspevok Algoritmu 18} \\ O((\ell(n))^3), \text{ príspevok algoritmu D} \end{array} \right\} O(n^2 + (\ell(n))^3)$$

čas

Chyba Algoritmu 19 závisí len od prvej časti, potom už je algoritmus deterministický. úspešnosť

$$\begin{aligned} Pr \left[\bigcup_{i=1}^{n-\ell(n)} E_i \right] &\geq \prod_{i=1}^{n-\ell(n)} \left(1 - \frac{2}{n-i+1}\right) = \\ &= \frac{\prod_{i=1}^{n-2} \left(1 - \frac{2}{n-i+1}\right)}{\prod_{j=n-\ell(n)+1}^{n-2} \left(1 - \frac{2}{n-j+1}\right)} = \frac{\frac{1}{\binom{n}{2}}}{\frac{1}{\binom{n}{2}}} = \frac{\binom{\ell(n)}{2}}{\binom{n}{2}} \end{aligned}$$

Ukázali sme nasledujúce tvrdenie

Lema 4.25 Ak $1 < \ell(n) < n$ je monotónna funkcia, tak Algoritmus 19 je polynomiálny pravdepodobnostný algoritmus, ktorý nájde minimálny rez s pravdepodobnosťou aspoň $\frac{\binom{\ell(n)}{2}}{\binom{n}{2}}$

Teraz poďme vhodne zvoliť $\ell(n)$. Podľa predchádzajúcej lemy 4.25 je pravdepodobnosť úspechu algoritmu DetRan aspoň

$$\frac{\binom{\ell(n)}{2}}{\binom{n}{2}} \leq \frac{\ell^2(n)}{n^2}$$

Ak teda spravíme $\frac{n^2}{\ell^2(n)}$ nezávislých opakovaní algoritmu DetRan(ℓ), dosiahneme v čas

$$O\left((n^2 + \ell^3(n)) \cdot \frac{n^2}{\ell^2(n)}\right) = O\left(\frac{n^4}{\ell^2(n)} + n^2 \ell(n)\right)$$

pravdepodobnosť úspechu aspoň

úspešnosť

$$1 - \left(1 - \frac{\binom{\ell(n)}{2}}{\binom{n}{2}}\right)^{\frac{n^2}{\ell^2(n)}} \geq 1 - \left(1 - \frac{\ell^2(n)}{n^2}\right)^{\frac{n^2}{\ell^2(n)}} = 1 - \frac{1}{e}$$

Z hľadiska časovej zložitosti je najlepšia voľba pre funkciu $\ell(n)$ taká, keď

$$\frac{n^4}{\ell^2(n)} = n^2 \ell(n) \implies \ell(n) = \lfloor n^{2/3} \rfloor$$

Zhrnutím týchto úvah máme

Veta 4.26 *Algoritmus Detran($n^{2/3}$) opakovaný $\frac{n^2}{\ell^2(n)} = n^{4/3}$ krát vypočíta v čase $O\left(\frac{n^4}{\ell^2(n)}\right) = O(n^{8/3})$ minimálny rez s pravdepodobnosťou úspechu aspoň $1 - \frac{1}{e}$.*

Získaný algoritmus⁴ je lepší ako najlepší známy deterministický. Skúsme vytvoriť ešte lepší algoritmus.

Vráťme sa k základnému algoritmu 18 a všimnime si, ako sa pri postupnosti kontrakcií pravdepodobnosť chyby mení.

$$\frac{2}{n}, \frac{2}{n-1}, \frac{2}{n-2}, \dots, \frac{2}{3}$$

Pravdepodobnosť chyby v priebehu opakovania kontrakcií rastie, preto budeme algoritmus častejšie opakovať ku koncu. Predstavme si, že jednotlivé opakované behy Algoritmu 18 robíme paralelne (obr. 4.1 vľavo). Vtedy si čas možno predstaviť ako plochu. Ak by sme behy vytvárali postupne (obr. 4.1 vpravo)—začneme s dvomi, po nejakom čase výpočtu každý proces pri náhodnej voľbe rozdelíme na dva, ..., bude čas odpovedať súčtu dĺžok ciest z koreňa do listov. Aj "opticky" to vyzerá lepšie. Ukážeme, že pri vhodnej voľbe deliacich bodov tomu tak naozaj je.

Počet behov zdvojnásobíme, keď sa počet vrcholov kontrakciami zredukoval na $1/\sqrt{2}$ -tinu. Keďže počet kontrakcií je $n - 2$, je počet behov, ktoré realizujeme, $2^{\log_{\sqrt{2}} n - 2} = O(n^2)$.

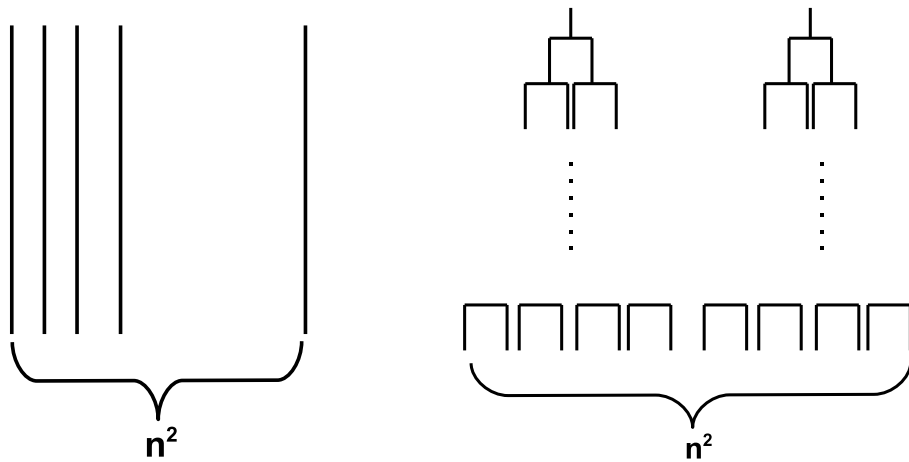
Algoritmus 20 RepTree(G)

- 1: ak $n \leq 6$ vypočítaj minimálny rez deterministicky
 - 2: $h \leftarrow \left\lceil 1 + \frac{n}{\sqrt{2}} \right\rceil$
 - 3: realizuj paralelne výpočet na dvoch behoch dotedy, kým kontrakciami nevzniknú grafy G/F_1 , G/F_2 veľkosti h
 - 4: RepTree(G/F_1)
 - 5: RepTree(G/F_2)
 - 6: return lepší minimálny rez
-

Veta 4.27 *Časová zložitost Algoritmu RepTree je $O(n^2 \log n)$, pravdepodobnosť úspechu aspoň $\frac{1}{\Omega(\log n)}$.*

Dôkaz: Začnime analýzou času.

⁴Kde je *random sampling*?



Obrázok 4.1: Opakovania Algoritmu Contract: vľavo n^2 kompletných behov, vpravo $O(n^2)$ behov, ktoré vznikajú postupným "delením"

- Veľkosť multigrafu klesá o (multiplikatívny) faktor $1/\sqrt{2}$, preto je hĺbka vno-
renia rekurzívnej funkcie najviac $\log_{\sqrt{2}} n = O(\log n)$.
- Časová zložitosť Algoritmu Contract, keď štartoval na n vrcholoch, je $O(n^2)$.
Prechod od m vrcholov k $\lceil 1 + \frac{m}{\sqrt{2}} \rceil$ teda môžeme zhora ohraničiť $O(m^2)$.
- Časovú zložitosť možno vyjadriť rekurentnou nerovnicou

$$T(n) \leq \begin{cases} O(1), & n \leq 6 \\ 2T\left(\lceil 1 + \frac{n}{\sqrt{2}} \rceil\right) + O(n^2) & \text{inak} \end{cases}$$

Využitím **Master theorem** dostaneme, že riešením je $O(n^2 \log n)$

Master theorem:

$$T(n) \leq \begin{cases} \Theta(1), & n = 1 \\ aT\left(\frac{n}{b}\right) + f(n) & \text{inak} \end{cases}$$

Potom pre riešenie platí

$$\begin{aligned} f(n) = O(n^{\log_b a - \epsilon}) &\Rightarrow T(n) = \Theta(n^{\log_b a}) \\ f(n) = \Theta(n^{\log_b a}) &\Rightarrow T(n) = \Theta(n^{\log_b a} \cdot \log n) \\ f(n) = \Omega(n^{\log_b a + \epsilon}) &\Rightarrow T(n) = \Theta(f(n)) \end{aligned}$$

Máme $\lceil 1 + \frac{n}{\sqrt{2}} \rceil \leq 2 + \frac{n}{\sqrt{2}} \approx \frac{n}{b}$, z čoho $b \approx \sqrt{2}$. Keďže $a = 2$, tak $n^2 = n^{\log_b a}$.

Pripomeňme si, že máme fixovaný minimálny rez C_{min} s cenou $|E(C_{min})| = k$ úspešnosť

- označme p_ℓ pravdepodobnosť toho, že graf G/F_i veľkosti $\lceil 1 + \frac{\ell}{\sqrt{2}} \rceil$ obsahuje C_{min} , ak ho obsahoval graf G veľkosti ℓ pred rozdelením na G/F_1 , G/F_2 .

$$p_\ell \geq \frac{\binom{\lceil 1 + \frac{\ell}{\sqrt{2}} \rceil}{2}}{\binom{\ell}{2}} = \frac{\left[1 + \frac{\ell}{\sqrt{2}}\right] \left(\left[1 + \frac{\ell}{\sqrt{2}}\right] - 1\right)}{\ell(\ell - 1)} \geq \frac{(\sqrt{2} + \ell)\ell}{2\ell(\ell - 1)} \geq \frac{1}{2}$$

- označme $Pr(n)$ pravdepodobnosť toho, že algoritmus RepTree nájde minimálny rez C_{min} .
- Predpokladajme, že na nejakej úrovni volania rekurzie máme graf H veľkosti ℓ , ktorý *obsahuje* minimálny rez veľkosti k . Nech H_1, H_2 sú tí potomci grafu H , v ktorých sa opäť aplikuje rekurzívne volanie (veľkosť grafu klesla z ℓ na $1 + \frac{\ell}{\sqrt{2}}$). Aká je pravdepodobnosť, že algoritmus nájde minimálny rez, ktorý bol obsiahnutý v H ? Minimálny rez sa (s pravdepodobnosťou p_ℓ) zachová pri prechode po "hrane" (H, H_1) (alebo (H, H_2)); predpokladáme (H, H_1) a následne rekurzívne volanie v H_1 s pravdepodobnosťou $Pr\left(\left[1 + \frac{\ell}{\sqrt{2}}\right]\right)$ dopočíta minimálny rez korektne. Preto $p_\ell Pr\left(\left[1 + \frac{\ell}{\sqrt{2}}\right]\right)$ je dolným odhadom na pravdepodobnosť toho, že z G/F vypočítame C_{min} redukciami na G/F_i , ak G/F ho obsahovalo
- pravdepodobnosť toho, že rekurziou *nevypočítame* C_{min} je nanajvyšš

$$\left(1 - p_\ell Pr\left(\left[1 + \frac{\ell}{\sqrt{2}}\right]\right)\right)^2$$

- $Pr(2) = 1$

$$\begin{aligned} Pr(\ell) &\geq 1 - \left(1 - p_\ell Pr\left(\left[1 + \frac{\ell}{\sqrt{2}}\right]\right)\right)^2 \geq 1 - \left(1 - \frac{1}{2} Pr\left(\left[1 + \frac{\ell}{\sqrt{2}}\right]\right)\right)^2 \\ &= Pr\left(\left[1 + \frac{\ell}{\sqrt{2}}\right]\right) - \frac{1}{4} \left(\left[1 + \frac{\ell}{\sqrt{2}}\right]\right)^2 \end{aligned}$$

Ukážeme, že riešením je $\Theta\left(\frac{1}{\log \ell}\right)$. Pomôžeme si tým, že sa riešenie Pr pozrieme optikou hĺbky rekurzie. Nech teda $k = \Theta(\log \ell)$ je hĺbka rekurzie. Potom dolný odhad na pravdepodobnosť úspechu $p(k)$ získame riešením rekurentnej rovnice

$$p(k) = \begin{cases} 1, & k=0 \\ p(k-1) - \frac{p(k-1)^2}{4}, & k > 0 \end{cases}$$

Označme $q(k) = 4/p(k) - 1$. Potom $p(k) = \frac{4}{q(k)+1}$

$$\begin{aligned} \frac{4}{q(k+1)+1} &= \frac{4}{q(k)+1} - \left(\frac{4}{q(k)+1}\right)^2 / 4 \\ \frac{1}{q(k+1)+1} &= \frac{q(k)}{(q(k)+1)^2} \end{aligned}$$

Úpravou dostávame $q(k+1) = q(k) + 1 + \frac{1}{q(k)}$ a následne:

$$\begin{aligned} q(k) &= q(k-1) + 1 + \frac{1}{q(k-1)} \\ &= q(k-2) + 1 + \frac{1}{q(k-2)} + 1 + \frac{1}{q(k-1)} \\ &= q(0) + k + \sum_{i=0}^{k-1} \frac{1}{q(i)} = 5 + k + \sum_{i=0}^{k-1} \frac{1}{q(i)} \end{aligned}$$

$$k < q(k) < k + 3 + \sum_{i=0}^{k-1} \frac{1}{i} = k + 3 + H_{k-1}$$

Teda $q(k) = k + \Theta \log k$, $p(k) = \frac{4}{q(k)+1} = \frac{4}{k + \Theta \log k} = \Theta\left(\frac{1}{k}\right)$ a teda $p(\ell) = \Theta\left(\frac{1}{\log \ell}\right)$

□

Na základe analýzy algoritmu RepTree je jasné, že $\log n$ nezávislých opakovaní tohto algoritmu stačí, aby sme s konštantnou pravdepodobnosťou dosiahli optimálne riešenie problému Min-Cut. Pri $O \log^2 n$ opakovaníach ide pravdepodobnosť neúspechu rýchlo k nule. Pritom čas v tomto prípade je $O(n^2 \log^3 n)$.

4.3.2 Opakovaná náhodná vzorka a 3-splniteľnosť

V tejto časti ukážeme, ako vhodné skombinovanie viacerých metód

- opakovanie
- náhodná vzorka
- lokálne prehľadávanie

vedie k "rozumnému" riešeniu NPÚ problému 3SAT. Doteraz nie je známy polytime pravdepodobnostný algoritmus na riešenie NPÚ problému s ohraničenou chybou, skôr sa predpokladá, že NP-ťažké sú ťažké aj pre polytime pravdepodobnostné. Čo ale môžeme dosiahnuť je rozumný exponenciálny čas.

f(n)	10	50	100	300
n!	$\approx 3.6 \cdot 10^6$	65 digits	158 digits	625 digits
2^n	1024	16 digits	31 digits	91 digits
$2^{n/2}$	32	$\approx 33 \cdot 10^6$	16 digits	46 digits
$(1.2)^n$	≈ 6.19	9100	$\approx 8.2 \cdot 10^7$	24 digits
$10 \cdot 2^{\sqrt{n}}$	≈ 30	≈ 1345	10240	$\approx 1.64 \cdot 10^6$
$n^2 \cdot 2^{\sqrt{n}}$	895	≈ 336158	$1.024 \cdot 10^7$	$\approx 1.48 \cdot 10^{11}$
n^6	10^6	$1.54 \cdot 10^{10}$	10^{12}	$\approx 7.29 \cdot 10^{14}$

Smerujeme k 1MC algoritmu pre 3KNF-splniteľnosť, ktorého časová zložitosť bude $O(|F| \cdot n^{3/2} \cdot (\frac{4}{3})^n)$. Základná idea spočíva v niekoľkonásobnom prehľadaní vhodného okolia náhodne vybraného vektora: cieľom je nájsť vektor, ktorý spĺňa vstupnú formulu.

Algoritmus 21 Schönning

```

1:  $N \leftarrow 0$  ▷ počet prezeraných vzoriek
2:  $S \leftarrow \lceil 20 \cdot \sqrt{3\pi n} \left(\frac{4}{3}\right)^n \rceil$  ▷ odhad počtu v tejto iterácii prezretých vzoriek
3: ▷ voľba neskôr vyplynie z analýzy pravdepodobnosti chyby
4:  $Found \leftarrow False$ 
5: while  $N < S$  and not Found do
6:    $N \leftarrow N + 1$ 
7:   náhodne zvol  $\alpha \in \{0, 1\}^n$ 
8:   if  $F(\alpha) = 1$  then  $Found \leftarrow True$ 
9:    $M \leftarrow 0$ 
10:  while  $M < 3n$  and not Found do
11:     $M \leftarrow M + 1$ 
12:    nájdí v  $F(\alpha)$  nesplnenú klauzulu  $C$  a náhodne zmeň niektorý z bitov
13:    v tejto klauzule, čím vznikne nová hodnota  $\alpha$ 
14:    if  $F(\alpha) = 1$  then  $Found \leftarrow True$ 
15: if  $Found = True$  then return (splniteľná,  $\alpha$ )
16: else return (nesplniteľná)

```

Veta 4.28 Algoritmus Schönning je 1MC pre problém splniteľnosti 3KNF formuly. Jeho časová zložitosť je $O(|F| \cdot n^{3/2} \cdot (4/3)^n)$

Dôkaz: Začneme časom.

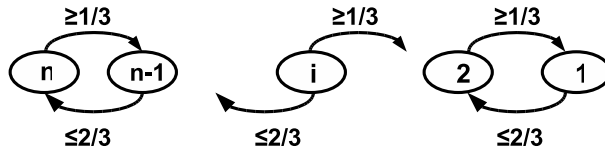
- pri použití opakovaného umocňovania vypočítame hodnotu S na riadku 2 v čase $O(n^2 \log n)$
- celkovo robíme S krát náhodnú vzorku s lokálnym vyhľadávaním

$$O(|F| \cdot \underbrace{n^{1/2}}_S \cdot (4/3)^n \cdot \underbrace{n}_M) = O(|F| \cdot n^{3/2} \cdot (4/3)^n)$$

Ak vstupná formula nie je splniteľná, algoritmus korektno odpovie, že je nespľniteľná. Chybu môže algoritmus spraviť len vtedy, ak sa mu nepodarí nájsť spĺňajúce priradenie, hoci formula splniteľná je. Spočítame, aká je pravdepodobnosť, že v prípade splniteľnej formuly nájdeme jedno *fixované* priradenie α^* . Označme

- p pravdepodobnosť toho, že jednou vzorkou a lokálnym prehľadávaním toto priradenie α^* nájdeme (riadky 7-14).
- $Dist(\alpha, \beta)$ - počet bitov, v ktorých sa vektory α, β líšia
- $Class(j) = \{\beta \in \{0, 1\}^n \mid Dist(\alpha^*, \beta) = j\}$; zrejme
 $Class(0) = \{\alpha^*\}$
 $|Class(j)| = \binom{n}{j}$

Pozrime sa na lokálne prehľadávanie ako na putovanie medzi jednotlivými triedami (obr.4.2)



Obrázok 4.2: Proces lokálneho prehľadávania možno vnímať ako putovanie po grafe. Cieľom je sa z náhodného vrchola dostať do vrchola 0.

- Pravdepodobnosť toho, že sa z j posunieme v jednom lokálnom kroku/jedným flipom do $(j-1)$ je aspoň $1/3$. Analogicky, presun z j do $(j+1)$ je s pravdepodobnosťou nanajvyš $2/3$.
- Označme $q_{j,i}$ pravdepodobnosť, že keď začneme v triede $Class(j)$, skončíme v α^* , pričom prejdeme *presne* $(j+i)$ krokov smerom ku α^* a i smerom od α^* . Zrejme $i, j \leq n$
- Popis lokálneho prehľadávania možno znázorniť reťazcom $\{+, -\}^{j+2i}$, pričom $j+2i \leq 3n$, $+$ znázorňuje posun smerom ku α^* , znak $-$ zas posun od α^* . Nie každý takýto reťazec je korektným popisom lokálneho prehľadávania. Ten korektný má v každom suffixe aspoň toľko znakov $+$ ako $-$. Dá sa ukázať, že takýchto reťazcov je

$$\binom{j+2i-1}{i} - \binom{j+2i-1}{i-1} = \binom{j+2i}{i} \frac{1}{j+2i} \quad (4.1)$$

Označme množinu týchto reťazcov \mathcal{B} .

- každé $w \in \mathcal{B}$ definuje množinu výpočtov $Event(w)$

$$Pr[Event(w)] \geq \left(\frac{1}{3}\right)^{j+i} \left(\frac{2}{3}\right)^i$$

Preto

$$q_{j,i} \geq \underbrace{\frac{1}{j+2i} \binom{j+2i}{i}}_{\#w} \underbrace{\left(\frac{1}{3}\right)^{j+i} \left(\frac{2}{3}\right)^i}_{\text{že je to podľa } w}$$

- q_j označuje pravdepodobnosť, že sa z $Class(j)$ dostaneme v rámci $3n$ krokov lokálneho prehľadávania do α^*

$$q_j \geq \sum_{i=0}^j q_{j,i} \quad 0 \leq i \leq j \leq n, \quad j + 2i \leq 3n$$

$$q_0 = 1$$

$$\begin{aligned} q_j &\geq \sum_{i=0}^j \left[\frac{1}{j+2i} \binom{j+2i}{i} \left(\frac{1}{3}\right)^{j+i} \left(\frac{2}{3}\right)^i \right] \\ &> j \cdot \frac{1}{3^j} \binom{3j}{j} \left(\frac{1}{3}\right)^{j+i} \left(\frac{2}{3}\right)^i = \frac{1}{3} \binom{3j}{j} \left(\frac{1}{3}\right)^{j+i} \left(\frac{2}{3}\right)^i \quad // i = j \text{ je dolný odhad} \\ &= \frac{1}{3} \cdot \frac{(3j)!}{j!(2j)!} \cdot \left(\frac{1}{3}\right)^{2j} \left(\frac{2}{3}\right)^j \approx \frac{1}{3} \frac{\sqrt{2\pi 3j} (3j/e)^{3j}}{\sqrt{2\pi j} (j/e)^j \sqrt{2\pi 2j} (2j/e)^{2j}} \left(\frac{1}{3}\right)^{2j} \left(\frac{2}{3}\right)^j \quad // r! \approx \sqrt{2\pi r} \left(\frac{r}{e}\right)^r \\ &= \frac{1}{2\sqrt{3\pi j}} \cdot \left(\frac{1}{2}\right)^j \end{aligned}$$

Na začiatku sme zvolili vektor náhodne, preto pravdepodobnosť p_j , že začneme v $Class(j)$, je

$$p(j) = \frac{\binom{n}{j}}{2^n}$$

A teraz už $p = Pr[\text{Schöning nájde po max } 3n \text{ krokoch}] \geq \sum_{j=0}^n p_j q_j$

$$\begin{aligned} p &> \sum_{j=0}^n \left[\left(\frac{1}{2}\right)^n \cdot \binom{n}{j} \cdot \frac{1}{2\sqrt{3\pi j}} \cdot \left(\frac{1}{2}\right)^j \right] \cdot 1^{n-j} = \sum_{j=0}^n \frac{1}{2\sqrt{3\pi j}} \left(\frac{1}{2}\right)^n \left(1 + \frac{1}{2}\right)^n \geq \\ &\geq \frac{1}{2\sqrt{3\pi n}} \left(\frac{1}{2}\right)^n \left(1 + \frac{1}{2}\right)^n = \frac{1}{2\sqrt{3\pi n}} \left(\frac{3}{4}\right)^n = \tilde{p} \end{aligned}$$

Preto pravdepodobnosť chyby po t opakovaníach je nanajvyšš

$$(1 - \tilde{p})^t \leq e^{-\tilde{p}t}$$

Ak teda vezmeme

$$t = S = 20\sqrt{3\pi n} \left(\frac{4}{3}\right)^n$$

dostaneme

$$Error(F) \leq (1 - \tilde{p})^t \leq e^{-\tilde{p}t} = e^{-10} < 5 \cdot 10^{-5}$$

Ostáva ešte ukázať platnosť (4.1). Nech $W(i, j)$ označuje počet "korektných" reťazcov dĺžky $i + 2j$, v ktorých je i núl a $j + i$ jednotiek. Indukciou argumentujeme, pár slov k (4.1)

že $W(i, j) = \binom{j+2i}{i} \frac{j}{j+2i}$

Pre $i = 0$ je $W(0, j) = 1$, pre $n = 1, 2$ je $i = 0$ a pre $n = 3$ je alebo $i = j = 0$ alebo $n \leq 3$ $i = 0, j = 3$. ✓

Pre $r + 2l < n$ IP platí. Pre $i + 2j = n$ rozlíšime situáciu podľa prvého symbolu $n > 3$

$$\text{prvý symbol je } \begin{cases} 1, & W(i, j-1) = \binom{j-1+2i}{i} \frac{j-1}{j-1+2i} \\ 0, & W(i-1, j+1) = \binom{j-1+2i}{i-1} \frac{j+1}{j-1+2i} \end{cases}$$

a dopočítame zvlášť pre $j = 1, j > 1$

□

4.3.3 Náhodná vzorka (random sampling) a generovanie kvadratických nezvyškov

Problém faktorizácie je ťažký. Máme polynomiálny algoritmus na testovanie prvočíselnosti, nemáme však polynomiálnu faktorizáciu. Ak za praktický algoritmus považujeme pravdepodobnostný polynomiálny s ohraničenou chybou, potom sa zdá, že čo je ťažké, je ťažké aj pre polynomiálne pravdepodobnostné algoritmy s ohraničenou chybou. Napriek tomu nejaké rozumné polynomiálne pravdepodobnostné algoritmy máme. Ukážeme Las Vegas pre kvadratické nezvyšky.

Definícia 4.5 *Majme pole Z_p , kde p je prvočíslo. Hovoríme, že $a \in Z_p$ je kvadratický zvyšok modulo p , ak $a \equiv x^2 \pmod{p}$. b je kvadratický nezvyšok modulo p , ak $\forall d \in Z_p \ d^2 \not\equiv b \pmod{p}$.*

Zatiaľ čo kvadratické zvyšky sa generujú ľahko, s nezvyškami to nie je také jednoduché; nemáme čas skúšať postupne všetko, kým ho nájdeme. Preto využijeme náhodnú vzorku. K aplikovaniu metódy potrebujeme existenciu

- efektívneho testu, ktorým overíme, či je dané číslo kvadratickým zvyškom alebo nie
- dostatočne veľa nezvyškov; ukážeme, že spomedzi čísel $1, 2, \dots, p-1$ je presne polovica zvyškov a polovica nezvyškov

Eulerovo kritérium

Veta 4.29 (Eulerovo kritérium) *Nech $p > 2$ je prvočíslo. $\forall a \in \{1, 2, \dots, p-1\}$*

1. *ak a je kvadratický zvyšok modulo p , potom $a^{(p-1)/2} \equiv 1 \pmod{p}$*
2. *ak a je kvadratický nezvyšok modulo p , potom $a^{(p-1)/2} \equiv p-1 \pmod{p}$*

Dôkaz: Využijeme Malú Fermátovu vetu:

$$\forall \text{ prvočíslo } p, a \in Z_p^* = \{d \in Z_p \mid \gcd(d, p) = 1\} \text{ platí } a^{p-1} \equiv 1 \pmod{p}$$

- p nepárne, väčšie ako 2, preto $\exists p' : p = 2p' + 1$

$$a^{(p-1)} - 1 = a^{(2p'+1-1)} - 1 = (a^{p'} - 1)(a^{p'} + 1) \equiv 0 \pmod{p}$$

Preto

$$\underbrace{a^{(p-1)/2} - 1 \equiv 0 \pmod{p} \text{ alebo } (a^{(p-1)/2} + 1) \equiv 0 \pmod{p}}_{a^{(p-1)/2} \in \{1, p-1\} \pmod{p}}$$

zvyšok

- Nech a je kvadratický zvyšok modulo p . Potom $a \equiv x^2 \pmod{p}$ a $a^{(p-1)/2} \equiv (x^2)^{(p-1)/2} \equiv x^{(p-1)} \equiv 1 \pmod{p}$

nezvyšok

- Nech a je kvadratický nezvyšok. Keďže $(Z_p^*, \cdot \pmod{p})$ je cyklická grupa, má generátor a a musí byť jeho *nepárna* mocnina; $a = g^{(2l+1)} \pmod{p}$

$$a^{(p-1)/2} \equiv \left(g^{(2l+1)}\right)^{(p-1)/2} \equiv g^{(p-1)} \cdot g^{(p-1)/2} \pmod{p}$$

Keďže $g^{p-1} \equiv 1 \pmod{p}$, je

$$g^{l(p-1)} \equiv g^{l(p-1)} \equiv 1^l \equiv 1 \pmod{p}$$

$$a^{(p-1)/2} \equiv 1 \cdot g^{(p-1)/2} \pmod{p}$$

Keďže g je generátor, je $g^{(p-1)/2} \not\equiv 1 \pmod{p}$ a vzhľadom k tomu, že platí $a^{(p-1)/2} \in \{1, p-1\} \pmod{p}$, je $a^{(p-1)/2} \equiv p-1 \pmod{p}$

□

Test na zistenie, či a je alebo nie je kvadratický zvyšok modulo p , je teda jednoduchý - výpočet $a^{(p-1)/2}$. Teraz sa zamyslime nad hustotou výskytu kvadratických nezvyškov.

Veta 4.30 *Ku každému prvočíslu $p \geq 3$ existuje presne $(p-1)/2$ nenulových prvkov v Z_p , ktoré sú kvadratický nezvyšok modulo p .*

Dôkaz: Uvažujme množinu $Quad(p) = \{1^2 \bmod p, 2^2 \bmod p, \dots, (p-1)^2 \bmod p\}$. Rovnosť $|Quad(p)| = (p-1)/2$ ukážeme dvomi nerovnosťami:

Pre $x \in \{1, 2, \dots, p-1\}$ platí ≤

$$(p-x)^2 = p^2 - 2px + x^2 = p(p-2x) + x^2 \equiv x^2 \pmod{p}$$

Stačí ukázať, že rovnica $x^2 = y^2 \pmod{p}$ má nanajvýš jedno riešenie. Wlog $y > x$, ≥
teda $y = x + i, i = 1, 2, \dots, p-2$

$$y^2 = x^2 + 2xi + i^2 \quad 2xi + i^2 \equiv 1 \pmod{p}$$

$i \neq 0 \Rightarrow 2x + i \equiv 0 \pmod{p}$ a teda $i = -2x$ je jediné riešenie □

Na základe vyššie povedaného je teda zrejmé, že opakovanie náhodného hľadania dovtedy, kým nejaký nezvyšok nenájde, je vlastne Las Vegas algoritmus, ktorý nikdy nepovie "neviem" a pre ktorý je pravdepodobnosť nekonečných výpočtov nulová.

4.4 Metóda svedkov

Pre použitie metódy svedkov je kľúčovým momentom dostatok svedkov. To úzko súvisí aj s odpoveďou na otázku, či pravdepodobnostné algoritmy môžu byť efektívnejšie ako deterministické. Ak máme efektívny pravdepodobnostný algoritmus založený na metóde svedkov a svedkov vieme efektívne generovať, máme vlastne aj efektívny deterministický algoritmus. Ak sú však svedkovia rozmiestnení náhodne,...

Pár slov na úvod

- Pre použitie metódy svedkov potrebujeme *dostatok svedkov*, čím väčšinou rozumieme to, že pomer kandidátov a svedkov je konštanta.
- Samotné aplikovanie metódy potom spočíva v náhodnom výbere kandidáta a overení, či je alebo nie je tento kandidát svedkom. Ak je v množine kandidátov dostatočne veľa svedkov, je pravdepodobnosť toho, že vybraný kandidát je svedok, dostatočná.

V tejto súvislosti sa budeme venovať prvočíselnosti, a to konkrétne

- testovaniu prvočíselnosti
- generovaniu náhodných prvočísel

Začneme opakovaním znenia ⁶ niektorých známych viet, ktoré budeme v argumentácii využívať.

Veta 4.31 (Malá Fermátová) $p \in PRIM, a \in Z_p^* = \{d \in Z_p | gcd(a, p) = 1\}$.
Potom

$$a^{(p-1)} \pmod{p} \equiv 1$$

⁶V tejto prednáške sú vety nástrojom, dôkazy preto vynechávame.

Veta 4.32 (O čínskych zvyškoch, II. verzia) *Nech $n = p \times q$, $\gcd(p, q) = 1$.
Nech $\oplus_{p,q}, \ominus_{p,q}$ sú operácie nad $Z_p \times Z_q$.*

$$\begin{aligned}(a_1, a_2) \oplus_{p,q} (b_1, b_2) &= ((a_1 \oplus_{p,q} b_1) \pmod p, (a_2 \oplus_{p,q} b_2) \pmod q) \\ (a_1, a_2) \ominus_{p,q} (b_1, b_2) &= ((a_1 \ominus_{p,q} b_1) \pmod p, (a_2 \ominus_{p,q} b_2) \pmod q)\end{aligned}$$

Potom $(Z_n, \oplus \pmod p, \ominus \pmod q)$ a $(Z_p \times Z_q, \oplus_{p,q}, \ominus_{p,q})$ sú izomorfné.

Veta 4.33 (O čínskych zvyškoch, I. verzia) *Nech $m = m_1 \times m_2 \times \dots \times m_k$,
kde $k \in N - \{0\}$, $\gcd(m_i, m_j) = 1$ pre $i \neq j$. Potom pre každú postupnosť $r_1 \in Z_{m_1}, \dots, r_k \in Z_{m_k}$ existuje jediné číslo $r \in Z_m : r \equiv r_i \pmod{m_i}$*

Veta 4.34 (Lagrange) *Pre každú podgrupu (H, \circ) konečnej grupy (A, \circ) platí*

$$|A| = \text{Index}_H(A) \cdot |H|$$

Tu

- $H \circ b = \{h \circ b | h \in H\}$
- $\text{Index}_H(A) = |\{H \circ b | b \in H\}|$

4.4.1 Hľadanie svedkov pre test prvočíselnosti

V tejto časti sa budeme venovať testu prvočíselnosti: pre vstupné n chceme vedieť, či je to prvočíslo alebo zložené číslo. Zaujímá nás pravdepodobnostný algoritmus, ktorého časová zložitosť je $O((\log n)^c)$ pre nízku konštantu c . Veľkosť konštanty c je dôležitá, keďže pre potreby kódovania sú prvočísla dlhé stovky cifier.

Jednotlivé algoritmy vychádzajú z vhodnej definície *svedka*.

naivný prístup

Za **naivný prístup** možno považovať test, ktorý vychádza zo základnej definície prvočísla ako čísla, ktoré je deliteľné iba jednotkou a samým sebou. Algoritmus je jednoduchý—testujeme postupne deliteľnosť n kandidátom a od 1 po $n - 1$, resp. \sqrt{n} . Zložitosť je $O(2^{\log n / 2})$, čo je vzhľadom na veľkosť vstupu $\log n$ exponenciálne.

požiadavky na svedka

- i. prvok a je svedkom pre zložené číslo, ak poskytuje efektívne overenie tohto faktu
- ii. ľahko rozlíšime, či kandidát je svedok alebo nie
- iii. množina kandidátov obsahuje dostatočne veľa svedkov

deliteľ

Pre identifikáciu zloženého čísla vieme triviálne využiť deliteľnosť: prvok a je svedok, že $n \notin \text{PRIM}^7 \Leftrightarrow a$ delí n

Takáto definícia spĺňa vlastnosti (i,ii), problém však môže byť s vlastnosťou (iii), ktorá nie je vždy splnená—v prípade $n = pq$, kde p, q sú prvočísla, je pravdepodobnosť, že náhodne zvolené číslo a je svedkom pre n iba $2/(n - 2)$.

Fermátova veta

Na základe malej Fermátovej vety máme nasledujúce kritérium — číslo a je svedok⁸ pre $n \notin \text{PRIM} \Leftrightarrow a^{\frac{n-1}{2}} \pmod n \neq 1$.

(ii) Efektívny výpočet $a^{p-1} \pmod p$ je založený na iterovanom umocňovaní

- $a^2 \pmod p = a \cdot a \pmod p$
 $a^{2^k} \pmod p = [(a^{2^{k-1}} \pmod p) \cdot (a^{2^{k-1}} \pmod p)] \pmod p$
- nech $b = \sum_{i=1}^k b_i 2^{i-1}$; potom

$$a^b = a^{b_1 2^0} \cdot a^{b_2 2^1} \cdot \dots \cdot a^{b_k 2^{k-1}}$$

⁷ PRIM označuje množinu prvočísel

$$\begin{aligned}
& a^b \text{ mod } p \text{ vypočítame:} \\
& - a_i = a^{2^{i-1}} \text{ mod } p \\
& - \prod_{i=1, b_i=1}^k a_i \text{ mod } p
\end{aligned}$$

Výpočet realizujeme $O(\log n)$ násobeniami nad Z_p , čo znamená $O((\log n)^3)$ bitových operácií. Takýto výpočet je teda efektívny, existuje však nekonečná množina tzv. Carmichaelových čísel, pre ktoré je tento test nevhodný. *Carmichael číslo* je také, že $\forall a \in \{1, \dots, n-1\}$ platí $a^{(n-1)} \text{ mod } n = 1$, ale n NIE JE prvočíslo.

Ďalší test vychádza z novej "definície" prvočísla

Veta 4.35 *Nech $p > 2$ je nepárne. Potom*

alternatívna definícia

$$p \text{ je prvočíslo} \Leftrightarrow a^{\frac{p-1}{2}} \text{ mod } p \in \{1, p-1\} \quad \forall a \in Z_p - \{0\}$$

Dôkaz: Nech p je nepárne prvočíslo, $p = 2p' + 1$. Z Malej Fermatovej vety dostávame \Rightarrow

$$\left. \begin{aligned}
a^{p-1} &\equiv 1 \pmod{p} \forall a \in Z_p - \{0\} \\
a^{p-1} &= a^{2p'} = (a^{p'} - 1)(a^{p'} + 1) + 1
\end{aligned} \right\} (a^{p'} - 1)(a^{p'} + 1) \equiv 0 \pmod{p}$$

Preto $a^{\frac{p-1}{2}} \text{ mod } p \in \{1, p-1\}$

Nech $a^{\frac{p-1}{2}} \in \{1, p-1\} \forall a \in Z_p - \{0\}$. Kvôli sporu predpokladajme, že $p = ab$. \Leftarrow

$$\left. \begin{aligned}
a^{\frac{p-1}{2}} \text{ mod } p &\in \{1, p-1\} \\
a^{\frac{p-1}{2}} \text{ mod } p &\in \{1, p-1\}
\end{aligned} \right\} (ab)^{\frac{p-1}{2}} \text{ mod } p = a^{\frac{p-1}{2}} \cdot b^{\frac{p-1}{2}} \text{ mod } p \in \{1, -1\}$$

Keďže $ab = p$,

$$0 = p \text{ mod } p = p^{\frac{p-1}{2}} \text{ mod } p = (ab)^{\frac{p-1}{2}} \text{ mod } p \in \{1, -1\} \text{ SPOR} \quad \square$$

Na základe tejto vety môžeme definovať nasledujúce kritérium pre svedka. Nech $n \geq 3$ je nepárne. Číslo $a \in \{1, \dots, n-1\}$ je svedok, že $n \notin \text{PRIM} \Leftrightarrow a^{\frac{n-1}{2}} \notin \{1, n-1\}$. Kritérium je vhodné pre nepárne n také, že $n \equiv 3 \pmod{4}$

Veta 4.36 *Pre každé $n, n \equiv 3 \pmod{4}$, platí*

- (a.) *ak n je prvočíslo, potom $a^{(n-1)/2} \text{ mod } n \in \{1, n-1\} \quad \forall a \in \{1, \dots, n-1\}$*
 (b.) *ak n je zložené, potom $a^{(n-1)/2} \text{ mod } n \notin \{1, n-1\}$ pre aspoň polovicu prvkov z $\underbrace{\{1, \dots, n-1\}}_{A(n-1)}$*

Dôkaz: Potrebujeme argumentovať (b). Vezmime teda *zložené* $n, n \equiv 3 \pmod{4}$.

Rozdelíme $A(n-1)$ na množinu svedkov a nesvedkov:

$$\begin{aligned}
Wit_n &= \{a \in A(n-1) \mid a^{(n-1)/2} \text{ mod } n \notin \{1, n-1\}\} \\
Euler &= \{a \in A(n-1) \mid a^{(n-1)/2} \text{ mod } n \in \{1, n-1\}\}
\end{aligned}$$

Ukážeme, že existuje *injektívne* zobrazenie z Euler do Wit_n , čo implikuje $|Euler| \leq |Wit_n|$.

Predpokladajme, že máme $b \in Wit_n$, pričom b^{-1} existuje (určíme neskôr). Pomocou *definícia* h_b b definujeme spomínané zobrazenie

$$h_b(a) = ab \text{ mod } n$$

Ukážeme, že h_b je zobrazenie Euler $\longrightarrow Wit_n$, je injektívne a napokon určíme b a b^{-1} .

Nech $a \in Euler$, $h_b(a) = ab \text{ mod } n$. Potom

$$h_b(a) \in Wit_n$$

$(a \cdot b)^{(n-1)/2} \pmod n = (a^{(n-1)/2} \pmod n) \cdot (b^{(n-1)/2} \pmod n) = \pm b^{(n-1)/2} \pmod n$
 Keďže $b \in \text{Wit}_n$, $b^{(n-1)/2} \pmod n \notin \{1, n-1\}$ a teda $h_b(a) \in \text{Wit}_n$.

Nech $a_1, a_2 \in \text{Euler}$, $a_1 \neq a_2$ a nech $h_b(a_1) = h_b(a_2)$, resp. $a_1 b \equiv a_2 b \pmod n$. *injekcia*
 Potom

$$a_1 \equiv a_1 b b^{-1} \pmod n \equiv a_2 b b^{-1} \pmod n = a_2$$

existencia b, b^{-1} Využime, že n je zložené⁹, $n = pq, \gcd(p, q) = 1$. Potom $(a \pmod p, a \pmod q)$ jednoznačne reprezentuje $a \in \{1, \dots, n-1\}$. Číslo a určíme jeho reprezentáciou. Kvôli tomu si všimnime, čo platí o reprezentácii čísel z Euler. Podľa definície pre $a \in \text{Euler}$ je $a^{(n-1)/2} \pmod{pq} \in \{1, n-1\}$

– ak $a^{(n-1)/2} = kpx + 1$, potom $a^{(n-1)/2} \pmod p \equiv a^{(n-1)/2} \pmod q \equiv 1$; reprezentácia $a^{(n-1)/2}$ je v tomto prípade $(1, 1)$

– ak $a^{(n-1)/2} = kpx + n - 1$, potom

$$a^{(n-1)/2} \pmod p = (n-1) \pmod p = (pq-1) \pmod p = p-1$$

$$a^{(n-1)/2} \pmod q = (n-1) \pmod q = (pq-1) \pmod q = q-1$$

V tomto prípade je $a^{(n-1)/2}$ reprezentované $(p-1, q-1)$

$b \in \text{Wit}_n$ Hodnotu b určíme reprezentáciou $(1, q-1)$. Overme, že $b \in \text{Wit}_n$:

$$(b^{(n-1)/2} \pmod p, b^{(n-1)/2} \pmod q) = (1^{(n-1)/2} \pmod p, (-1)^{(n-1)/2} \pmod q) = (1, -1).$$

Číslo $b \notin \text{Euler}$, preto $b \in \text{Wit}_n$.

$b^{-1} = b$ Inverzný prvok k b je b :

$$b \cdot b = (1, q-1) \odot (1, q-1) = (1 \pmod p, (q-1)(q-1) \pmod q) = (1, 1) \quad \square$$

Algoritmus 22 SSSA - zjednodušený Solovay-Strassen

- 1: zvol' náhodne $a \in \{1, \dots, n-1\}$
 - 2: $A \leftarrow a^{(n-1)/2} \pmod n$
 - 3: **if** $A \in \{1, -1\}$ **then** return($n \in \text{PRIM}$)
 - 4: **else** return($n \notin \text{PRIM}$)
-

Na základe predchádzajúcich úvah dostávame, že pre zložené n je pravdepodobnosť toho, že výstupom algoritmu SSSA je korektná odpoveď $n \notin \text{PRIM}$, aspoň polovica.

Veta 4.37 *Pre $n = 4k + 3$ je algoritmus SSSA polytime 1MC pre zložené čísla.*

4.4.2 Algoritmus Solovay-Strassen pre testovanie prvočíselnosti

V tejto časti sa zameriame na odstránenie podmienky $n \equiv 3 \pmod 4$. Pridáme ďalšie kritérium, ktoré pomáha identifikovať zložené čísla – $\gcd \neq 1$.

Definícia 4.6 $a \in \{1, \dots, n-1\}$ je *svedok*, že nepárne $n \notin \text{PRIM}$, ak

- $\gcd(a, n) > 1$ alebo
- $\gcd(a, n) = 1$ a $a^{(n-1)/2} \pmod n \notin \{1, -1\}$

Kritérium vieme efektívne overiť, keďže $\gcd(a, n)$ vieme efektívne počítať pomocou Euclidovho algoritmu.

Rekurzívne volanie znižuje aspoň jeden z parametrov na polovicu, preto je hĺbka rekurzie $O(\log b)$, pri "školskom" algoritme delenia je celková zložitosť $O((\log a + b)^3)$.

Hoci je podmienka z definície 4.6 vhodná pre $n \not\equiv 3 \pmod 4$, pre Carmichaelove čísla vhodná nie je. Platí ale

⁹ Doriešte pre $n = p^i$

Algoritmus 23 Euclid(a,b)

-
- ```

1: if $b = 0$ then return(a)
2: else return(Euclid(b , $a \bmod b$))

```
- 

**Fakt 4.38** *Nech  $n$  je zložené, nie Carmichaelovo. Potom aspoň polovica zo  $Z_n - \{0\}$  potvrdzuje  $n \notin PRIM$  podľa podmienky z definície 4.6*

Pokračujeme vo vylepšovaní definície svedka.

**Definícia 4.7** *Nech  $p > 2$  je prvočíslo,  $a$  kladné,  $\gcd(a, p) = 1$ . Legendrov symbol pre  $a$  a  $p$  je*

$$\text{Leg} \left[ \frac{a}{p} \right] \begin{cases} 1, & a \text{ je kvadratický zvyšok modulo } p \\ -1, & a \text{ je kvadratický nezvyšok modulo } p \end{cases}$$

Pre prvočíslo  $p$  a  $a$  s  $\gcd(a, p) = 1$  platí

$$\text{Leg} \left[ \frac{a}{p} \right] = a^{(p-1)/2} \pmod{p}$$

Na základe tohto faktu vieme  $\text{Leg} \left[ \frac{a}{p} \right]$  efektívne počítať.

**Definícia 4.8** *Nech  $n = p_1^{k_1} p_2^{k_2} \dots p_\ell^{k_\ell}$  je faktorizácia nepárneho  $n \geq 3$ , pričom  $k_j$  sú kladné. Ak  $\gcd(a, n) = 1$ , potom Jacobiho symbol pre  $a, n$  je*

$$\text{Jac} \left[ \frac{a}{n} \right] = \prod_{i=1}^{\ell} \left( \text{Leg} \left[ \frac{a}{p_i} \right] \right)^{k_i} = \prod_{i=1}^{\ell} \left( a^{(p_i-1)/2} \pmod{p_i} \right)^{k_i}$$

**Fakt 4.39** *Pre každé  $a, n$  spĺňajúce podmienku definície Jacobiána*

$$\text{Jac} \left[ \frac{a}{n} \right] \in \{-1, 1\}$$

Jacobián chceme využiť na definíciu svedka, faktorizáciu ale nemáme. Na základe vlastností z nasledujúcej lemy 4.40 ho budeme vedieť efektívne vypočítať.

**Lema 4.40** *Majme nepárne  $n \geq 3$ ,  $a, b : \gcd(a, n) = \gcd(b, n) = 1$ . Platí*

1.  $\text{Jac} \left[ \frac{ab}{n} \right] = \text{Jac} \left[ \frac{a}{n} \right] \cdot \text{Jac} \left[ \frac{b}{n} \right]$
2.  $\text{Jac} \left[ \frac{a}{n} \right] = \text{Jac} \left[ \frac{b}{n} \right]$  pre  $a \equiv b \pmod{n}$
3.  $\text{Jac} \left[ \frac{a}{n} \right] = -1^{\frac{(a-1)(n-1)}{2}} \text{Jac} \left[ \frac{n}{a} \right]$  pre  $n$  nepárne
4.  $\text{Jac} \left[ \frac{1}{n} \right] = 1$ ,  $\text{Jac} \left[ \frac{n-1}{n} \right] = (-1)^{(n-1)/2}$
5.  $\text{Jac} \left[ \frac{2}{n} \right] = \begin{cases} -1, & \text{pre } n \bmod 8 \in \{3, 5\} \\ 1, & \text{pre } n \bmod 8 \in \{1, 7\} \end{cases}$

**Dôkaz:** Ukážeme prvé dve vlastnosti, ostatné ako cvičenie.

$$\begin{aligned} \text{Jac} \left[ \frac{ab}{n} \right] &= \prod_{i=1}^{\ell} \left( (ab)^{(p_i-1)/2} \pmod{p_i} \right)^{k_i} = \\ &= \prod_{i=1}^{\ell} \left( a^{(p_i-1)/2} \pmod{p_i} \right)^{k_i} \left( b^{(p_i-1)/2} \pmod{p_i} \right)^{k_i} = \\ &= \prod_{i=1}^{\ell} \left( a^{(p_i-1)/2} \pmod{p_i} \right)^{k_i} \prod_{i=1}^{\ell} \left( b^{(p_i-1)/2} \pmod{p_i} \right)^{k_i} = \\ &= \text{Jac} \left[ \frac{a}{n} \right] \text{Jac} \left[ \frac{b}{n} \right] \end{aligned}$$

Stačí, ak ukážeme, že

$$(a \equiv b \pmod{p}) \& (gcd(a, b) = 1) \Rightarrow Leg \left[ \frac{a}{p} \right] = Leg \left[ \frac{b}{p} \right]$$

$$\begin{aligned} a \equiv b \pmod{p} &\Rightarrow a = pr + z \\ & b = ps + z \end{aligned}$$

$$\begin{aligned} Leg \left[ \frac{a}{p} \right] &= a^{(p-1)/2} \pmod{p} = (pr + z)^{(p-1)/2} \pmod{p} = \\ &= \sum_{i=0}^{(p-1)/2} \binom{(p-1)/2}{i} (pr)^{\frac{p-1}{2}-i} z^i \pmod{p} = z^{(p-1)/2} \pmod{p} \end{aligned}$$

$$\text{Analogicky } Leg \left[ \frac{b}{p} \right] = z^{(p-1)/2} \pmod{p} \quad \square$$

A teraz už spomínaný algoritmus na výpočet Jacobiána. Každé rekurzívne volanie

---

**Algoritmus 24** Jacobian(a,n)
 

---

▷ vstupom sú nepárne  $n \geq 3$  a  $a$ , pričom  $gcd(a, n) = 1$

case

$$\begin{aligned} a = 1 &: Jac \left[ \frac{1}{n} \right] \leftarrow 1 \\ a = 2 \& n \pmod{8} \in \{1, 7\} &: Jac \left[ \frac{2}{n} \right] \leftarrow 1 \\ a = 2 \& n \pmod{8} \in \{3, 5\} &: Jac \left[ \frac{2}{n} \right] \leftarrow -1 \\ a \text{ nepárne} &: Jac \left[ \frac{a}{n} \right] \leftarrow Jac \left[ \frac{2}{n} \right] Jac \left[ \frac{a/2}{n} \right] \\ a > n &: Jac \left[ \frac{a}{n} \right] \leftarrow Jac \left[ \frac{a \pmod{n}}{n} \right] \\ \text{inak} &: Jac \left[ \frac{a}{n} \right] \leftarrow (-1)^{\frac{a-1}{2} \frac{n-1}{2}} Jac \left[ \frac{n \pmod{a}}{a} \right] \end{aligned}$$


---

zniži aspoň jeden z parametrov aspoň na polovicu, Preto je hĺbka rekurzcie  $O(\log n)$ . Manipulácia s parametrami je  $O(1)$  aritmetických, resp.  $O((\log a + n)^2)$  bitových operácií. Čas výpočtu  $Jac \left[ \frac{a}{n} \right]$  je  $O((\log(a + n))^3)$  binárnych operácií.

A teraz sme už pripravení definovať novú charakterizáciu svedka.

*Jac-svedok*

**Definícia 4.9** *Nech  $n \geq 3$  je nepárne. Hovoríme, že  $a \in \{1, \dots, n-1\}$  je Jac-svedok pre  $n \notin PRIM$ , ak*

- $gcd(a, n) \neq 1$ , alebo
- $gcd(a, n) = 1$  a  $Jac \left[ \frac{a}{n} \right] \neq a^{(n-1)/2} \pmod{n}$

O korektnosti pravdepodobnostného algoritmu založeného na Jac-svedkoch hovorí nasledujúca veta.

**Veta 4.41** *Pre každé nepárne  $n \geq 3$  platí*

1. *ak  $n$  je prvočíslo, potom  $Jac \left[ \frac{a}{n} \right] = Leg \left[ \frac{a}{n} \right] = a^{(n-1)/2} \pmod{n}$  pre všetky  $a \in \{1, 2, \dots, n-1\}$*
2. *ak  $n$  je zložené číslo, potom  $Jac \left[ \frac{a}{n} \right] \neq a^{(n-1)/2} \pmod{n}$  pre aspoň polovicu z tých  $a \in \{1, 2, \dots, n-1\}$ , pre ktoré  $gcd(a, n) = 1$*

**Dôkaz:** Dokazovať potrebujeme vlastnosť 2. Majme teda  $n \geq 3$  nepárne. Množina kandidátov je  $Z_n - \{0\}$ . Opäť uvažujme dve množiny

$$\begin{aligned} \overline{Wit}_n &= \{a \in Z_n^* \mid Jac \left[ \frac{a}{n} \right] = a^{(n-1)/2} \pmod{n}\} \\ Wit_n &= Z_n^* - \overline{Wit}_n \end{aligned}$$

Ukážeme, že  $|\overline{Wit}_n| \leq |Z_n^*|/2$ , z čoho potom dostaneme  $|\{1, 2, \dots, n-1\} - \overline{Wit}_n| \geq |\overline{Wit}_n|$ . Pri argumentácii si pomôže vlastnosťami grúp a podgrúp. Začneme tým,

že ukážeme, že  $(\overline{\text{Wit}}_n, \odot_{\text{mod}n})$  je vlastná podgrupa  $(Z_n^*, \odot_{\text{mod}n})$ — $\overline{\text{Wit}}_n$  je grupa a existuje prvok zo  $Z_n - \overline{\text{Wit}}_n$ .

grupa

Nech  $a, b \in \overline{\text{Wit}}_n$ . Potom

$$\left. \begin{aligned} \text{Jac} \left[ \frac{ab}{n} \right] &= \text{Jac} \left[ \frac{a}{n} \right] \text{Jac} \left[ \frac{b}{n} \right] = \left( a^{\frac{n-1}{2}} \pmod{n} \right) \left( b^{\frac{n-1}{2}} \pmod{n} \right) \\ &= (ab)^{\frac{n-1}{2}} \pmod{n} \end{aligned} \right\} \Rightarrow ab \in \overline{\text{Wit}}_n$$

Potrebuje ukázať existenciu prvku  $a$  zo  $Z_n - \overline{\text{Wit}}_n$ . Nech  $n = \underbrace{p_1^{i_1}}_q \underbrace{p_2^{i_2} \dots p_k^{i_k}}_m, \geq 1$ , *vlastná podgrupa*

je faktorizácia  $n$ . Prvok  $a$  nebudme hľadať priamo v  $Z_n^*$ , ale v  $Z_q \times Z_m$ . Nech  $g$  je generátor cyklickej grupy  $(Z_q^*, \odot_{\text{mod}q})$ . Prvok  $a$  zvolíme tak, že

$$\left. \begin{aligned} a &\equiv g \pmod{q} \\ a &\equiv 1 \pmod{m} \end{aligned} \right\} a \stackrel{\text{def}}{=} (g, 1) \in Z_q \times Z_m$$

Ak  $m = 1$ , ako  $a$  vezmeme  $g$ .

Fakt, že  $\text{gcd}(a, n) = 1 \Leftrightarrow$  žiadne z prvočísel  $p_1, \dots, p_k$  nedelí  $a$ .

$a \in Z_n^*$

Ak by  $p_1$  delilo  $a$ , tak  $a = 0 \pmod{p_1}$  súčasne s  $g \equiv a \pmod{p_1}$  je v spore s tým, že  $g$  je generátor.

Ak  $p_r$  delí  $a$ , potom

$$\left. \begin{aligned} a &= p_r b \\ a &= mx + 1 \end{aligned} \right\} p_r b = mx + 1 = p_r \left( \frac{m}{p_r} \right) x + 1$$

Dostali sme, že  $p_r$  delí 1, pričom  $p_r > 1$ .

Dôkaz tejto časti rozdelíme na dve časti podľa hodnoty  $i_1$  ( $1, \geq 2$ ). Vypočítame  $a \notin \overline{\text{Wit}}_n$   $\text{Jac} \left[ \frac{a}{n} \right], a^{(n-1)/2} \pmod{n}$  a porovnáme.

**$i_1 = 1, n = p_1 m, \text{gcd}(p_1, m) = 1$**

$$\begin{aligned} \text{Jac} \left[ \frac{a}{n} \right] &= \text{Jac} \left[ \frac{a}{p_1} \right] \prod_{j=2}^k \left( \text{Jac} \left[ \frac{a}{p_j} \right] \right)^{i_j} \\ &\stackrel{a \equiv 1 \pmod{m}}{=} \text{Jac} \left[ \frac{a}{p_1} \right] \prod_{j=2}^k \underbrace{\left( \text{Jac} \left[ \frac{1}{p_j} \right] \right)^{i_j}}_{=1} \\ &= \text{Jac} \left[ \frac{a}{p_1} \right] = \text{Jac} \left[ \frac{a}{p_1} \right] = \text{Leg} \left[ \frac{a}{p_1} \right] \\ &= -1 \quad // \text{generátor nemôže byť kvadratický zvyšok} \end{aligned}$$

Teraz hodnota  $a^{(n-1)/2} \pmod{n}, n = p_1 m$

$$a^{(n-1)/2} \pmod{m} = (a \pmod{m})^{(n-1)/2} \pmod{m} = 1^{(n-1)/2} \pmod{m} = 1$$

Preto  $a^{(n-1)/2} \pmod{n} = 1$  a  $\boxed{-1 = \text{Jac} \left[ \frac{a}{n} \right] \neq a^{(n-1)/2} \pmod{n} \text{ pre } i_1 = 1}$

**$i_1 \geq 2$**

Nech by  $a \in \overline{\text{Wit}}_n$ , čo by znamenalo  $a^{(n-1)/2} \pmod{n} = \text{Jac} \left[ \frac{a}{n} \right] \in \{1, -1\}$ . Preto  $a^{(n-1)} \equiv 1 \pmod{n}$ . Keďže  $g \equiv a \pmod{q}$ ,  $n = p_1^{i_1} m = qm$

$$1 = a^{n-1} \pmod{n} = a^{n-1} \pmod{q} = (a \pmod{q})^{n-1} \pmod{q} = g^{n-1} \pmod{q}$$

$g$  je generátor cyklickej grupy, jeho rád je  $|Z_q^*|$ , preto  $|Z_q^*|$  delí  $n - 1$

$Z_q^* = \{x \in Z_q \mid p_1 \text{ nedelí } x\}$ .

$$|Z_q^*| = |Z_q| - \underbrace{\frac{|Z_q|}{p_1}}_{\text{deliteľné } p_1} = p_1(p_1^{i_1-1} - p_1^{i_2-1})$$

Máme teda

$$\left. \begin{array}{l} p_1 \text{ delí } |Z_q^*|, |Z_q^*| \text{ delí } n-1 \\ p_1 \text{ delí } n = p_1^{i_1} m \end{array} \right\} \implies p_1 \text{ delí } n, n-1 \quad \square$$

---

**Algoritmus 25** SSA- Solovay-Strassen
 

---

▷ vstupom je nepárne  $n \geq 3$

- 1: vygeneruj náhodne  $a \in \{1, 2, \dots, n-1\}$
  - 2: vypočítaj  $\gcd(a, n)$  ▷  $O((\log n)^3)$
  - 3: **if**  $\gcd(a, n) \neq 1$  **then** return( $n \notin \text{PRIM}$ )
  - 4:  $J \leftarrow \text{Jac} \left[ \frac{a}{n} \right]; A \leftarrow a^{(n-1)/2} \pmod n$  ▷  $O((\log n)^3)$
  - 5: **if**  $J = A$  **then** return( $n \in \text{PRIM}$ )
  - 6: **elsereturn**( $n \notin \text{PRIM}$ )
- 

**Veta 4.42** Algoritmus Solovay-Strassen je polytime 1MC algoritmus pre rozpoznávanie zložených čísel.

**Dôkaz:** Korektnosť dôkazu vyplýva z predchádzajúcich úvah.

čas

Máme  $a < n$ , teda na reprezentáciu  $a, n$  stačí  $\lceil \log n \rceil + 1$  bitov.

- 2 na výpočet  $\gcd$  použijeme Euclidov algoritmus, ktorý zbehne v čase  $O((\log n)^3)$  (bitové operácie)
- 4 na výpočet hodnôt  $J, A$  stačí  $O((\log n)^3)$  bitových operácií
- 5 porovnanie realizujeme v  $O(\log n)$

úspech

$n \in \text{PRIM}$  nenájdeme svedka, ktorý by potvrdil, že  $n$  je zložené

$n \notin \text{PRIM}$  pravdepodobnosť, že svedka nájdeme, je podľa vety 4.41 aspoň  $1/2$ .

□

### 4.4.3 Generovanie náhodných prvočísel

V tejto časti sa budeme zaoberať generovaním prvočísel; našou úlohou je pre danú dĺžku  $\ell$  vygenerovať náhodné prvočíslo dĺžky  $\ell$ . Stratégia je zrejmá — vygenerujeme náhodné číslo  $n$  príslušnej dĺžky a overíme, či je prvočíslo. Pre zvýšenie pravdepodobnosti úspechu spravíme niekoľko ( $2\ell^2$ ) generovaní  $n$  a niekoľko ( $k$ ) testov prvočíselnosti pre príslušné  $n$ . Na testovanie prvočíselnosti používame algoritmus Solovay-Strassen.

**Veta 4.43** Algoritmus PrimGen je algoritmus s ohraničenou chybou, ktorý v čase polynomiálnom od dĺžky (výstupu)  $\ell$  vygeneruje prvočíslo.

**Algoritmus 26** PrimGen( $\ell, k$ )

---

▷ vstupom je dĺžka prvočísla  $\ell$  a počet opakovaní testov  $k$

- 1:  $Found \leftarrow False, I \leftarrow 0$
- 2: **while**  $not Found$  and  $I < 2\ell^2$  **do**
- 3:   vygeneruj náhodnú postupnosť  $a_1, a_2, \dots, a_{\ell-2}$  bitov
- 4:  $n \leftarrow 2^{\ell-1} + \sum_{i=1}^{\ell-2} a_i 2^i + 1$
- 5:   sprav  $k$  nezávislých testov Solovay-Strassenovým algoritmom
- 6:   **if** niektorý dokázal  $n \notin \text{PRIM}$  **then**  $I \leftarrow I + 1$
- 7:   **else Found**  $\leftarrow True$ , return( $n$ )
- 8: **if**  $I = 2\ell^2$  **then** return(nenašiel)

---

**Dôkaz:** Keďže v prípade úspechu je dĺžka výstupu exponenciálna od dĺžky vstupu, meriame zložitosť algoritmu vzhľadom na veľkosť výstupu.

*čas*  $2\ell^2$  opakovaní while-cyklu,  $k$  opakovaní SSA testu, ktorý trvá  $O(\ell^3)$  dáva celkovú zložitosť bitových operácií  $O(\ell^5 k)$ , resp. pre  $k = \ell$   $O(\ell^6)$ .

*úspešnosť* Algoritmus je neúspešný z dvoch dôvodov

I ak sa mu nepodarilo vygenerovať prvočíсло (resp. číslo, o ktorom by dokázal, že je zložené)

II na výstup dal ako prvočíсло číslo, ktoré je v skutočnosti zložené

Aby sme na výstup nedali žiadne číslo, musel každý test, ktorý sme robili, dopadnúť "zle":  $n \notin \text{PRIM}$

- pravdepodobnosť, že náhodne zvolené číslo  $n$  dĺžky  $\ell$

JE prvočíсло, je aspoň  $\frac{1}{\ln n} > \frac{1}{2\ell}$

NIE JE prvočíсло je nanajvýš  $(1 - \frac{1}{2\ell})$

- pravdepodobnosť, že  $\ell$  behov Solovay-Strassenovho algoritmu uspeje v dôkaze, že zložené  $n \notin \text{PRIM}$  je  $w_\ell \geq 1 - 1/2^\ell$

- $Prob[PrimGen(\ell, \ell) = \text{"nenašiel"}] < ((1 - \frac{1}{2\ell}) w_\ell)^{2\ell^2} < (1 - \frac{1}{2\ell})^{2\ell^2} < e^{-\ell}$

Pre  $\ell \geq 100$  je  $e^{-\ell} \ll 10^{-40}$

Označme  $p_i$  pravdepodobnosť toho, že v  $i$ -tom behu,  $i \in 1, \dots, 2\ell^2$ , dáme na výstup ako prvočíсло číslo, ktoré je v skutočnosti zložené. To znamená, že  $(i-1)$  predchádzajúcich behov vygenerovalo zložené číslo a úspešne to aj overilo, ale v poslednom behu vygenerované číslo sa nepodarilo dokázať.

- $p_1 < (1 - \frac{1}{2\ell}) \cdot \frac{1}{2^\ell}$

- $p_i \leq [(1 - \frac{1}{2\ell}) w_\ell]^{i-1} (1 - \frac{1}{2\ell}) \cdot \frac{1}{2^\ell} = (1 - \frac{1}{2\ell})^i w_\ell \frac{1}{2^\ell}$

$$\begin{aligned}
Pr[\text{chyba}(\ell, \ell)] &\leq p_1 + \sum_{j=2}^{2\ell^2} p_j \leq \left(1 - \frac{1}{2\ell}\right) \cdot \frac{1}{2^\ell} + \sum_{j=2}^{2\ell^2} \left(1 - \frac{1}{2\ell}\right)^j w_\ell \frac{1}{2^\ell} \leq \\
&\leq \left(1 - \frac{1}{2\ell}\right) \cdot \frac{1}{2^\ell} \left[ \sum_{j=1}^{2\ell^2-1} \left(1 - \frac{1}{2\ell}\right)^j w_\ell + 1 \right] \\
&\leq \left(1 - \frac{1}{2\ell}\right) \cdot \frac{1}{2^\ell} \left[ \sum_{j=1}^{2\ell^2-1} \underbrace{\left(1 - \frac{1}{2\ell}\right)^j}_{<2} + 1 \right] \\
&< \left(1 - \frac{1}{2\ell}\right) \cdot \frac{1}{2^\ell} 2\ell^2 \leq \frac{\ell^2}{2^{\ell-1}}
\end{aligned}$$

Pre  $\ell \geq 100$  je  $\frac{\ell^2}{2^{\ell-1}} \leq 1.58 \cdot 10^{-26}$

□

## 4.5 Optimalizačné úlohy a náhodné zaokrúhľovanie

Užitočným nástrojom pri riešení optimalizačných úloh je lineárne programovanie (LP). Problém lineárneho programovania je definovaný nasledovne:

LP

### Definícia 4.10

vstup matica  $A = [a_{ij}], i = 1, \dots, m; j = 1, \dots, n$

vektor  $b \in \mathbb{R}^m$

$c \in \mathbb{R}^n$

ohraničenia  $M(A, b, c) = \{X \in (\mathbb{R}^{0,+})^n \mid AX = b\}$

cena  $\text{cost}(X, (A, b, c)) = c^T X = \sum_{i=1}^n c_i x_i$

cieľ minimalizácia

Resp. ohraničenia s rovnosťou nahradíme ohraničeniami s nerovnosťami:

$$\begin{aligned}
\text{ohraničenia} \quad &\sum_{i=1}^n a_{ji} x_i = b_j, j \in I_1 \\
&\sum_{i=1}^n a_{ji} x_i \geq b_s, s \in I_2 \\
&\sum_{i=1}^n a_{ji} x_i \leq b_r, r \in \{1, \dots, m\} - (I_1 \cup I_2)
\end{aligned}$$

O úlohe lineárneho programovania vieme, že efektívnosť jej riešenia závisí od oboru hodnôt riešenia  $X$ .

- pre  $X \in \mathbb{R}$  existuje algoritmus polynomiálnej zložitosti
- pri  $X \in \mathbb{Z}$ , tzv. ILP, resp.  $X \in \{0, 1\}^n$ , tzv. 0-1LP algoritmus polynomiálnej zložitosti nepoznáme

Vzhľadom k tejto rôznej zložitosti využívame LP pri návrhu *aproximačných* algoritmov. Základná schéma využitia je nasledovná:

1. formulácia pôvodného problému ako úlohy ILP, resp. 0-1LP;  $ILP(I)$
2. relaxácia na úlohu lineárneho programovania LP;  $Rel - LP(I)$
3. transformácia optimálneho riešenia relaxovaného problému na *prípustné* riešenie pôvodného problému. Výsledná kvalita získaného algoritmu zrejme závisí od tejto transformácie. Jednou z vhodných metód je *metóda náhodného zaokrúhľovania*.

Spravme relaxáciu  $ILP(I) \rightsquigarrow Rel - LP(I)$ . Nech

$\alpha$  je optimálne riešenie pre  $Rel - LP(I)$

$Opt_U(I)$  je optimálne riešenie  $ILP(I)$ .

Potom (prečo?)

$$cost(\alpha) = Opt(Rel - LP(I)) \begin{cases} \leq Opt_U(I), & \text{pre minimalizačný problém} \\ \geq Opt_U(I), & \text{pre maximalizačný problém} \end{cases}$$

Pri návrate od optimálneho  $\alpha$  k prípustnému  $\beta$  sa snažíme poukázať kvalitu  $\alpha$  čo najmenej. Toto je tá časť, ktorá súvisí s NP-ťažkosťou pôvodného problému ILP.

Na príklade Min-VC ukážeme formuláciu optimalizačných úloh ako úloh LP.

**Príklad 4.4** Uvažujme problém minimálneho vrcholového pokrytia, v ktorom hľadáme minimálnu množinu vrcholov  $U$ , ktorá pokrýva všetky hrany. Prípustným riešením je teda vektor  $(x_1, \dots, x_n) \in \{0, 1\}^n$  taký, že  $x_i = 1 \Leftrightarrow v_i \in U$ . Formulujme ako úlohu LP: Min-VC

**minimalizovať**  $\sum_{i=1}^n x_i$

**podmienky**  $x_i + x_j \geq 1 \quad \forall (v_i, v_j) \in E$   
 $x_i \in \{0, 1\}$

**relaxácia** podmienku  $x_i \in \{0, 1\}$  nahradzame dvomi podmienkami  $x_i \geq 0$  a  $x_i \leq 1$

Majme optimálne riešenie  $\alpha = (\alpha_1, \dots, \alpha_n) \in [0, 1]^n$  relaxovaného problému. K nemu zostrojíme prípustné riešenie  $\beta$  pôvodného problému zaokrúhlením takto

$$\beta_i = 1 \Leftrightarrow \alpha_i \geq 1/2$$

Ukážeme, že sme získali 2-aproximačný algoritmus.

- V optimálnom riešení relaxovanej úlohy  $\alpha$  platí  $\alpha_i + \alpha_j \geq 1$  pre každú hranu  $(v_i, v_j) \in E$ . To ale znamená, že aspoň pre jedno  $\alpha_t, t \in \{i, j\}$  platí  $\alpha_t \geq 1/2$ . Následne do vrcholového pokrytia zaradíme vrchol  $v_t$ . Každá hrana je teda pokrytá. prípustné riešenie
- Keďže  $\beta_i \leq 2\alpha_i$  pre každé  $i$ , o cene získaného prípustného riešenia  $\beta$  platí kvalita

$$cost(\beta) = \sum_{i=1}^n \beta_i \leq 2 \sum_{i=1}^n \alpha_i = 2cost(\alpha)$$

$$\text{Aproximačný pomer je } \frac{cost(\beta)}{cost(\alpha)} \leq \frac{2cost(\alpha)}{cost(\alpha)} = 2$$

◇

### 4.5.1 Náhodné zaokrúhľovanie a problém MaxSAT

Vráťme sa opäť k riešeniu problému MaxSat. V časti 2.1 sme ukázali, že náhodne vygenerované priradenie hodnôt premenným vedie v očakávanom prípade k splneniu polovice klauzúl. V tejto časti ukážeme efektívny pravdepodobnostný algoritmus, ktorý je výsledkom kombinácie metódy relaxácie k LP a náhodného zaokrúhľovania. Majme teda

$$F(x_1, \dots, x_n) = F_1 \wedge F_2 \wedge \dots \wedge F_m, \quad \text{kde } F_i \text{ je elementárna disjunkcia}$$

Označme

$Set^+(F_i)$  množina premenných zo  $Set(F_i)$ , ktoré sa v  $F_i$  vyskytujú bez negácie

$Set^-(F_i)$  množina premenných zo  $Set(F_i)$ , ktoré sa v  $F_i$  vyskytujú s negáciou

$In^+(F_i)$  indexy premenných z  $Set^+(F_i)$

$In^-(F_i)$  indexy premenných z  $Set^-(F_i)$

Riešime úlohu

*formulácia*

**maximalizovať**  $\sum_{j=1}^m Z_j$

**lineárne ohraničenia**  $\sum_{i \in In^+(F_j)} y_i + \sum_{i \in In^-(F_j)} (1 - y_i) \geq Z_j, j = 1, \dots, m$

**n+m nelineárnych ohraničení**

$$y_i \in \{0, 1\} \quad i = 1, \dots, n$$

$$Z_j \in \{0, 1\} \quad j = 1, \dots, m$$

**relaxácia**

$$1 \geq y_i, y_i \geq 0$$

$$1 \geq Z_j, Z_j \geq 0$$

Označme  $\alpha(u), u \in \{y_1, \dots, y_n, Z_1, \dots, Z_m\}$  hodnotu  $u$  v optimálnom riešení relaxovaného problému. Ľahko vidno, že  $\sum_{j=1}^m \alpha(Z_j)$  je horný odhad na počet klauzúl, ktoré môžu byť splnené (prečo?). Kľúčovým pre kvalitné prípustné riešenie je zakokrúhľovanie; spravíme ho náhodne:

---

**Algoritmus 27 RRR** - random rounding

---

1: vstupom je úloha 0/1 LP(F)

2: relaxácia 0/1-LP(F)  $\rightsquigarrow$  Rel-LP(F)

3: nech  $(\alpha(y_1), \dots, \alpha(y_n), \alpha(Z_1), \dots, \alpha(Z_m))$  je optimálne riešenie Rel-LP(F)

4: vygeneruj náhodne  $\gamma_1, \dots, \gamma_n \in [0, 1]^n$

5: **if**  $\gamma_i \in [0, \alpha(y_i)]$  **then**  $\beta_i \leftarrow 1$

6: **else**  $\beta_i \leftarrow 0$

7: return  $(\beta_1, \dots, \beta_n)$

---

Uvedomme si, že takto definovaná  $\alpha(y_i)$  je vlastne pravdepodobnosť toho, že  $\beta_i = 1$ <sup>10</sup>

**Lema 4.44** *Nech  $k \in N, F_j$  je klauzula s  $k$  literálmi,  $(\alpha(y_1), \dots, \alpha(y_n), \alpha(Z_1), \dots, \alpha(Z_m))$  je optimálne riešenie Rel-LP(F). Potom pravdepodobnosť, že  $\beta = RRR(F)$  spĺňa  $F_j$ , je aspoň*

$$\left(1 - \left(1 - \frac{1}{k}\right)^k\right) \alpha(Z_j)$$

**Dôkaz:** Keďže sa na ostatné klauzuly formuly nepozerať, môžeme predpokladať, že  $F_j = x_1 \vee \dots \vee x_k$ . Z podmienok LP vyplýva, že

$$x_1 + x_2 + \dots + x_k \geq Z_j$$

Klauzula  $F_j$  bude *nesplnená* práve vtedy ak každé  $\beta_i \leftarrow 0, i = 1, \dots, k$ . To nastane s pravdepodobnosťou  $\prod_{i=1}^k (1 - \alpha(y_i))$ .  $F_j$  bude splnená s pravdepodobnosťou

$$1 - \prod_{i=1}^k (1 - \alpha(y_i))$$

---

<sup>10</sup>na rozdiel od 1/2 v prípade náhodného generovania pravdivostných hodnôt

čo nadobúda minimum pre  $\alpha(y_i) = \frac{\alpha(Z_j)}{k}$ . Dostávame teda

$$Pr[F_j(\beta) = 1] \geq 1 - \prod_{i=1}^k \left(1 - \frac{\alpha(Z_j)}{k}\right)$$

čo je vlastne funkcia jednej premennej— $\alpha(Z_j)$ . Využijeme nasledujúci fakt, ktorého platnosť ukončuje dôkaz.

**Fakt 4.45** *Nech  $k \in \mathbb{N}$ . Potom*

$$f_k(Z) = 1 - \left(1 - \frac{r}{k}\right)^k \geq \left(1 - \left(1 - \frac{1}{k}\right)^k\right) r = g_k(r) \quad \forall r \in [0, 1]$$

K dôkazu faktu 4.45 si stačí uvedomiť, že

- funkcia  $g_r$  je lineárna
- funkcia  $f_k$  je konkávna
- $f_k(0) = 0 = g_k(0)$
- $f_k(1) = 1 - \left(1 - \frac{1}{k}\right)^k = g_k(1)$

□

Teraz už ľahko zanalyzujeme Algoritmus 27.

**Veta 4.46** *Algoritmus RRR je polynomiálny*

1. pravdepodobnostný  $E\left[\frac{e}{e-1}\right]$ -aproximačný pre problém MaxSAT
2. pravdepodobnostný  $E\left[\frac{k^k}{k^k - (k-1)^k}\right]$ -aproximačný pre problém Max-EkSAT<sup>11</sup>

**Dôkaz:** Začnime s analýzou času. Redukcia 0/1-LP na Rel-LP je v lineárnom čase, optimálne riešenie Rel-LP vypočítame v polynomiálnom čase a získanie prípustného riešenia pre 0/1-LP z optimálneho pre Rel-LP spravíme v lineárnom čase. čas

Zopakujme si – algoritmus  $A$  je  $E[\Delta]$ -aproximačný, ak  $E\left[\frac{Opt(U)}{A(U)}\right] \leq \Delta$ , resp. ekvivalentne  $E[A] \geq Opt/\Delta$ . Stačí teda ukázať, že očakávaný počet splnených klauzúl je aspoň  $\frac{e-1}{e} \sum_{j=1}^m \alpha(Z_j)$ , resp.  $\frac{k^k - (k-1)^k}{k^k} \sum_{j=1}^m \alpha(Z_j)$ . chyba

Nech  $\delta \in \{0, 1\}^n$  je potenciálne riešenie. Potom

$$Prob[\delta = (\delta_1, \dots, \delta_n)] = \prod_{i=1}^n q_i, \quad q_i = \begin{cases} \alpha(y_i), & \text{ak } \delta_i = 1 \\ (1 - \alpha(y_i)), & \text{ak } \delta_i = 0 \end{cases}$$

Uvažujme indikačnú premennú  $Z_j$

$$Z_j(\delta) = \begin{cases} 1, & \text{ak } \delta \text{ spĺňa } F_j \\ 0, & \text{ak } \delta \text{ } F_j \text{ nespĺňa} \end{cases}$$

Potom  $E[Z_j]$  je pravdepodobnosť toho, že RRR(F) spĺňa  $F_j$ .

$$E[Z_j] \geq \left(1 - \left(1 - \frac{1}{k}\right)^k\right) \alpha(Z_j)$$

<sup>11</sup>Problém Max-EkSAT je MaxSAT obmedzený na vstupy, v ktorých každá klauzula má presne  $k$  literálov.

Nás zaujíma  $E[Z]$  pre  $Z = \sum_{j=1}^m Z_j$

$$E[Z] = \sum_{j=1}^m E[Z_j] \geq \sum_{j=1}^m \left(1 - \left(1 - \frac{1}{k}\right)^k\right) \alpha(Z_j) = \left(1 - \left(1 - \frac{1}{k}\right)^k\right) \sum_{j=1}^m \alpha(Z_j)$$

*EkSAT*

$$\begin{aligned} E[\text{Ratio} - \text{EkSAT}] &\leq \frac{\text{Opt}_{\text{EkSAT}}(F)}{E[Z]} \leq \frac{\sum_{j=1}^m \alpha(Z_j)}{\left(1 - \left(1 - \frac{1}{k}\right)^k\right) \sum_{j=1}^m \alpha(Z_j)} \\ &= \frac{1}{\left(1 - \frac{(k-1)^k}{k^k}\right)} = \frac{k^k}{k^k - (k-1)^k} \end{aligned}$$

*SAT*

Keďže  $\left(1 - \frac{1}{k}\right)^k \leq e^{-1}$ , je  $1 - \left(1 - \frac{1}{k}\right)^k \geq 1 - e^{-1} = \left(1 - \frac{1}{e}\right)$ . Preto

$$\begin{aligned} E[Z] &\geq \left(1 - \frac{1}{e}\right) \sum_{j=1}^m \alpha(Z_j) \\ E[\text{Ratio} - \text{SAT}] &= \frac{\text{Opt}_{\text{MaxSAT}}(F)}{E[Z]} \leq \frac{1}{\left(1 - \frac{1}{e}\right)} = \frac{e}{e-1} \end{aligned}$$

Pri vylepšovaní opakovaním opakujeme len časť od vygenerovania náhodného vektora  $\gamma$ .  $\square$

#### 4.5.2 Náhodná vzorka s náhodným zaokrúhľovaním

Prezentovali sme dva algoritmy pre riešenie problému MaxSAT — RSAM s pomerom 2 a RRR s pomerom  $\frac{e}{e-1}$ . Hoci  $2 > \frac{e}{e-1}$ ,  $\frac{2^k}{2^k-1} < \frac{k^k}{k^k-(k-1)^k}$  a teda pre dlhé klauzuly je naivný RSAM lepší. V skutočnosti sa dá ukázať (zamyslite sa), že existujú také vstupy  $I, I', I''$  že

- $E[\text{RSAM}(I)] > E[\text{RRR}(I)]$  //MaxSAT
- $E[\text{RRR}(I')] > E[\text{RSAM}(I')]$  //MaxSAT
- $E[\text{Ratio} - \text{RRR}(I'')] < E[\text{Ratio} - \text{RSAM}(I'')]$  //Max-EkSAT

Keďže majú rôzne správanie, je rozumné pokúsiť sa ich skombinovať. Spustíme oba algoritmy nezávisle a výsledkom bude lepší zo získaných výsledkov: Poďme analy-

---

#### Algoritmus 28 COMB-MaxSAT

---

- 1:  $\beta \leftarrow \text{RSAM}(F)$
  - 2:  $\gamma \leftarrow \text{RRR}(F)$
  - 3: **if**  $\beta$  spĺňa viac klauzúl ako  $\gamma$  **then** return  $\beta$
  - 4: **else** return  $\gamma$
- 

*čas*

zovať tento algoritmus. Keďže oba algoritmy su polynomiálne, je polynomiálny aj COMB-MaxSAT.

*aproximačný pomer*

Na vstupe máme formulu  $F(x_1, \dots, x_n) = F_1 \wedge \dots \wedge F_m$ .

- |                                                        |                                                                                               |
|--------------------------------------------------------|-----------------------------------------------------------------------------------------------|
| $(S_{\text{RSAM}}, \text{Prob}_S)$                     | $2^n$ možných výpočtov                                                                        |
| $(S_{\text{RRR}}, \text{Prob}_R)$                      | priestor určuje optimálne riešenie $\alpha$ pre Rel-LP(F)                                     |
| $(S_{\text{RSAM}} \times S_{\text{RRR}}, \text{Prob})$ | pričom $\text{Prob}[C, D] \stackrel{\text{def.}}{=} \text{Prob}_S[C] \times \text{Prob}_R[D]$ |

Majme beh algoritmu COMB-MaxSAT, ktorý vypočítal optimálne priradenia  $\beta, \gamma$ . Označme

$Y[\beta, \gamma]$  počet splnených klauzúl v  $F(\beta)$

$Z[\beta, \gamma]$  počet splnených klauzúl v  $F(\gamma)$

$U[\beta, \gamma] = \max\{Y[\beta, \gamma], Z[\beta, \gamma]\}$ . Keďže maximum nie je menšie ako aritmetický priemer

$$E[U] \geq \frac{E[Y] + E[Z]}{2}$$

Kvôli podrobnejšej analýze si klauzuly rozdelíme do triedy podľa počtu literálov;  $C_k$  budú tie klauzuly, ktoré majú presne  $k$  literálov.

Algoritmus RRR vypočíta priradenie, ktoré spĺňa nanajvyš  $\sum_{j=1}^m \alpha(Z_j)$  klauzúl.

$$\begin{aligned} E[Z] &\geq \sum_{k \geq 1} \sum_{F_j \in C(k)} \left(1 - \left(1 - \frac{1}{k}\right)^k\right) \alpha(Z_j) \\ E[Y] &= \sum_{k \geq 1} \sum_{F_j \in C(k)} \left(1 - \frac{1}{2^k}\right) \geq \sum_{k \geq 1} \sum_{F_j \in C(k)} \left(1 - \frac{1}{2^k}\right) \alpha(Z_j) \\ E[U] &\geq \frac{E[Z] + E[Y]}{2} \geq \frac{1}{2} \sum_{k \geq 1} \sum_{F_j \in C(k)} \left[ \left(1 - \frac{1}{2^k}\right) + \left(1 - \left(1 - \frac{1}{k}\right)^k\right) \right] \alpha(Z_j) \\ &\geq \frac{1}{2} \cdot \frac{3}{2} \sum_{k \geq 1} \sum_{F_j \in C(k)} \alpha(Z_j) = \frac{3}{4} \sum_{j=1}^m \alpha(Z_j) \end{aligned}$$

Dokázali sme teda nasledovnú vetu:

**Veta 4.47** Algoritmus COMB-MaxSAT je polynomiálny pravdepodobnostný  $E[4/3]$ -aproximačný algoritmus pre problém MaxSAT.

## 4.6 Semidefinitné programovanie a problém MaxCut

V tejto časti využijeme náhodné zaokrúhľovanie pri hľadaní maximálneho hranového rezu. Na rozdiel od predchádzajúcej časti bude problém naformulovaný ako problém semidefinitného programovania.

Máme neorientovaný graf  $G = (E, V)$  a váhovaciu funkciu  $w$ , ktorá každej hrane  $MaxCut$  priradí racionálnu váhu;  $w : E \rightarrow Q^+$ . Hľadáme hranový rez  $(S, \bar{S})$  maximálnej váhy

$$\max \sum_{e \in S \times \bar{S}} w(e)$$

Využijeme existenciu polynomiálneho algoritmu na riešenie tzv. semidefinitného programovania:

- $Y$  je  $n \times n$  matica s reálnymi hodnotami
- chceme maximalizovať lineárnu funkciu jednotlivých  $y_{i,j}$   
+ lineárne podmienky na  $y_{i,j}$   
+  $Y$  je symetrická, pozitívne semidefinitná  $\rightsquigarrow \forall X \in \mathbb{R}^n \ X^T Y X \geq 0$   
(má vlastné čísla a  $n$  lineárne nezávislých vlastných vektorov)

Nech  $A \in \mathbb{R}^{n \times n}$

- $trace(A) = tr(A)$  = súčet štvorcov diagonálnych prvkov

$$\bullet A \circ B = \text{tr}(A^T B) = \sum_{i=1}^n \sum_{j=1}^n a_{i,j} b_{i,j}$$

Nech  $C, D_1, \dots, D_k \in M_n$  sú symetrické reálne matice typu  $n \times n$  a  $d_1, \dots, d_k \in \mathbb{R}$  *semidefinitný program*  
 Problém  $S$  semidefinitného programovania je formulovaný nasledovne

$$S: \begin{array}{ll} \max & C \circ Y \\ \text{ohraničenia} & D_i \circ Y = d_i \quad 1 \leq i \leq k \\ & Y \geq 0 \quad (Y \text{ pozitívne semidefinitná}) \\ & Y \in M_n \end{array}$$

Ak  $C, D_1, \dots, D_k$  sú diagonálne, máme lineárne programovanie.

*Prípustným riešením* je taká matica  $A$ , ktorá spĺňa všetky podmienky  $S$ . Ak  $A$  nie je prípustné riešenie, potom  $C \in \mathbb{R}^{n \times n} : C \circ Y \leq b$  je *separujúca nadrovina* pre  $A$ .

Semidefinitný program vieme v rámci aditívnej chyby  $\varepsilon$  riešiť v čase  $\text{poly}(n, \log(1/\varepsilon))$

**Veta 4.48** *Nech  $S$  je semidefinitný program,  $A \in \mathbb{R}^{n \times n}$ . V polynomiálnom čase vieme overiť, či  $A$  je prípustným riešením pre  $S$ . Ak nie je, nájdeme v tomto čase separujúcu nadrovinu.*

**Dôkaz:** Ak  $A$  nie je prípustné

- $A$  nie je symetrická  $a_{i,j} > a_{j,i}$ , potom  $y_{i,j} \leq y_{j,i}$  je tá nadrovina
- $A$  nie je pozitívne semidefinitná, potom má záporné vlastné číslo  $\lambda$  a príslušný vlastný vektor  $v$ . Potom  $(v \cdot v^T) \circ Y = v^T \cdot Y \cdot v \geq 0$  je tá hyperrovina
- $A$  nespĺňa niektorú z ohraničujúcich podmienok. Vtedy každá taká nesplnená podmienka je tou hľadanou hyperrovinou

□

*kvadratický program*

Optimalizácia kvadratickej funkcie s celočíselnými hodnotami pri kvadratických ohraničeniach

*vektorový program*

Majme  $n$  vektorových premenných  $\tilde{v}_1, \dots, \tilde{v}_n \in \mathbb{R}^n$ . Vektorovým programom nazveme optimalizáciu lineárnej funkcie skalárnych súčinov  $\tilde{v}_i \cdot \tilde{v}_j$  pri lineárnych ohraničeniach vzhľadom k  $\tilde{v}_i \cdot \tilde{v}_j$

Jednou metódou riešenia je transformácia striktno kvadratického programu na vektorový. Aplikujeme na problém MaxCut.

$$\begin{array}{ll} \text{celočíselná premenná} & \rightsquigarrow \text{vektor} \\ x \cdot y & \rightsquigarrow \text{príslušný skalárny súčin} \end{array}$$

MaxCut

**strict  $S$**

$y_i = \mp 1$  indikačná premenná pre vektor  $v_i$   
 $(S, \bar{S}) = (\{v_i | y_i = 1\} \{v_i | y_i = -1\})$

**vektorový  $v$**

vektorové premenné  $\tilde{v}_1, \dots, \tilde{v}_n$

$y_i, y_j$  v rôznych množinách  $\Rightarrow y_i y_j = -1$   
 v jednej množine  $\Rightarrow y_i y_j = 1$

$$\max \quad \frac{1}{2} \sum_{1 \leq i < j \leq n} w_{i,j} (1 - y_i y_j)$$

$$\text{podm.} \quad \begin{array}{ll} y_i^2 = 1 & v_i \in V \\ y_i \in Z & v_i \in V \end{array}$$

$$\max \quad \frac{1}{2} \sum_{1 \leq i < j \leq n} w_{i,j} (1 - \tilde{v}_i \cdot \tilde{v}_j)$$

$$\text{podm.} \quad \begin{array}{ll} \tilde{v}_i \cdot \tilde{v}_j = 1 & v_i \in V \\ \tilde{v}_i \in \mathbb{R}^n & v_i \in V \end{array}$$

Uvedomme si, že vektory  $\tilde{v}_i$  ležia na jednotkovej guli  $S_{n-1}$ . Ak je riešenie striktného programu  $y_i$ , tak príslušný vektor  $\tilde{v}_i = (y_i, 0, \dots, 0)$ .

**Lema 4.49** *Vektorový problém  $v$  je ekvivalentný semidefinitnému programu  $S$*

**Dôkaz:** Nech  $a_1, \dots, a_n$  je optimálne riešenie pre vektorový problém  $v$ . Vezmime  $\Rightarrow$  maticu  $W$  takú, že jej stĺpce sú  $a_1, \dots, a_n$ . Potom riešením striktného programu je matica  $A = WW^T$ .

Majme riešenie semidefinitného programu  $A$ . Potom existuje matica  $W$  tak, že  $\Leftarrow$   $A = W^T W$ . Stĺpce sú riešením vektorového programu.  $\square$

Aplikujme tento prístup na riešenie problému MaxCut. Majme optimálne riešenie  $a_1, \dots, a_n$  vektorovej verzie problému  $v$ ;  $OPT_v$  nech je jeho hodnota. Potrebujeme zdefinovať "zaokrúhlenie" – prechod k celočíselnému prípustnému riešeniu.

Označme  $\Theta_{i,j}$  uhol medzi  $a_i$  a  $a_j$ . Príspevok týchto dvoch vektorov do vektorového  $\Theta_{i,j}$  riešenia je<sup>12</sup>

$$\frac{w_{i,j}}{2} \underbrace{(1 - \cos \Theta_{i,j})}_{(*)}$$

$$\left. \begin{array}{l} \Theta_{i,j} = 0 \quad \rightsquigarrow (*) = 0 \\ \Theta_{i,j} = \pi/2 \quad \rightsquigarrow (*) = 1 \end{array} \right\} \text{separovať budeme tie vektory, ktoré zvierajú veľký uhol}$$

K náhodnému zaokrúhľovaniu potrebujeme náhodné vektory na jednotkovej guľi/sfére. Tie môžeme získať napríklad takto:

**Lema 4.50** *Nech  $x_1, \dots, x_n$  sú náhodné z normálneho rozdelenia s mean=0, odchylka=1. Nech  $d = \frac{\sqrt{x_1^2 + \dots + x_n^2}}{2}$ . Potom vektor  $(x_1/d, \dots, x_n/d)$  je náhodný vektor na jednotkovej guľi  $S_{n-1}$ .*

A teraz sa vráťme k problému MaxCut

- zvolíme náhodne vektor  $r \in S_{n-1}$
- $S = \{v_i \mid a_i \cdot r \geq 0\}$

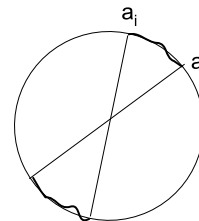
*náhodné zaokrúhľovanie*

**Lema 4.51** *Pre vyššie definovaný spôsob zaokrúhľovania platí*

$$Pr[v_i, v_j \text{ sú separované}] = \frac{\Theta_{i,j}}{\pi}$$

**Dôkaz:**

- vezmime rovinu, v ktorej ležia  $a_i, a_j$  a spravme priemet  $r$  do tejto roviny
- $r$  separuje  $a_i, a_j$  ak sa premietne do vyznačených oblúkov
- $\frac{2\Theta_{i,j}}{2\pi} = \frac{\Theta_{i,j}}{\pi}$



$\square$

Všimnime si kvalitu Algoritmu MaxCut. Nech

$W$  je premenná, ktorá počíta váhu podľa Algoritmu MaxCut

---

<sup>12</sup> $\cos \alpha = \frac{\sum a_i b_i}{\|a\| \|b\|}$ , kde  $\|a\| = \|b\| = 1$

**Algoritmus 29** MaxCut

- 
- 1: vyrieš vektorový program
  - 2: zvol náhodne  $r$  na guli  $S_{n-1}$
  - 3:  $S = \{v_i \mid a_i \cdot r \geq 0\}$
  - 4: return  $S$
- 

$$\alpha = \frac{2}{\pi} \underbrace{\min_{0 \leq \Theta \leq \pi} \frac{\Theta}{1 - \cos \Theta}}_m$$

Potom

$$\pi = \frac{2m}{\alpha} \quad \text{a} \quad \frac{\Theta}{\pi} = \frac{\Theta\alpha}{2m} \geq \frac{\alpha(1 - \cos \Theta)}{2}$$

$$\begin{aligned} E[W] &= \sum_{1 \leq i < j \leq n} w_{i,j} Pr[v_i, v_j \text{ separované}] = \sum_{1 \leq i < j \leq n} w_{i,j} \frac{\Theta_{i,j}}{\pi} \\ &\geq \alpha \sum_{1 \leq i < j \leq n} \frac{w_{i,j}}{2} (1 - \cos \Theta_{i,j}) = \alpha OPT_v \end{aligned}$$

Dokázali sme  $E[W] \geq \alpha OPT_v$ . Keďže  $\alpha > 0.87856$ , analýza algoritmu MaxCut vedie k nasledujúcemu tvrdeniu.

**Veta 4.52** *Existuje pravdepodobnostný aproximačný algoritmus pre problém MaxCut s aproximačným pomerom 0.87856*

**Dôkaz:**

$$\begin{aligned} \text{Nech } T &= \sum_{e \in E} w(e) \\ a \text{ také, že } E[W] &= aT \\ p &= Pr[W < (1 - \varepsilon)aT] \end{aligned}$$

Vieme, že  $W \leq T$ . Preto  $aT \leq p(1 - \varepsilon)aT + (1 - p)T$ , z čoho úpravou dostaneme

$$p \leq \frac{1-a}{1-a+\varepsilon a}$$

Z druhej strany zas

$$T \geq E[W] = aT \geq \alpha OPT_v \geq \alpha OPT \geq \frac{\alpha T}{2}$$

Posledná nerovnosť vychádza z analýzy jednoduchého greedy algoritmu. Po vykrátení  $T$  máme odhad na parameter  $a$ ,  $1 \geq a \geq \frac{\alpha}{2}$ , ktorý ďalej využijeme.

$$p \leq \frac{1-a}{1-a+a\varepsilon} = 1 - \frac{a\varepsilon}{1-a+a\varepsilon} \leq 1 - \frac{\varepsilon \frac{\alpha}{2}}{1 - \frac{\alpha}{2} + \varepsilon} = 1 - c$$

Ak teda algoritmus MaxCut zopakujeme  $1/c$  krát, ako výsledok vezmeme najlepší z jednotlivých výsledkov, dostaneme rez s váhou  $W'$ , pričom platí

- $Pr[W' \geq (1 - \varepsilon)aT] \geq 1 - (1 - c)^{1/c} \geq 1 - e^{-1}$
- $aT \geq \alpha OPT > 0.87856 OPT$
- Vieme zvoliť  $\varepsilon$  tak, aby  $(1 - \varepsilon)aT \geq 0.87856 OPT$

□

# Kapitola 5

## Ďalšie príklady pravdepodobnostných algoritmov

### 5.1 Paralelné a distribuované výpočty

V tejto časti sa budeme venovať pravdepodobnostným algoritmom pre niektoré problémy z oblasti paralelných a distribuovaných výpočtov. Začneme veľmi hrubým pohľadom na to, čo pre tieto účely rozumieme pod pojmom paralelný a distribuovaný výpočet

#### Paralelné výpočty

- procesory sú úzko späté: podobné, skoro rovnaká rýchlosť, výmena informácie v skoro nulovom čase
- na konci výpočtu všetci skončia a spoločne "vlastnia" riešenie

#### Distribuované výpočty

- nevieme nič o rýchlostiach, zdržaniach,..
- vyžadujeme, aby výpočet niekedy skončil, garantujeme korektnosť výsledku a počítame počet prenášaných správ

Základným modelom paralelných výpočtov je PRAM:

- globálnu pamäť tvorí  $M$  buniek
- každý procesor má lokálnu pamäť malej/resp. konštantnej veľkosti
- paralelné synchronizované kroky sú "trojkroky"
  - read z globálnej pamäte
  - lokálny výpočet
  - zápis do globálnej pamäte
- riešenie konfliktov nepripúšťa súčasný zápis - exclusive write, teda modely EREW, CREW
- uvažujeme polynomiálny počet procesorov a polylogaritmickej čas; dostávame tak zložitostné triedy NC a RNC

PRAM

**Definícia 5.1** *Trieda RNC je trieda jazykov , pre ktoré existuje PRAM A taký, že  $RNC, ZNC \forall x \in \Sigma^*$*

- $x \in L \Rightarrow Pr[A(x) \text{ je akceptuj}] \geq 1/2$
- $x \notin L \Rightarrow Pr[A(x) \text{ je akceptuj}] = 0$
- *plynmiálny počet procesorov*
- *polylogaritmický čas*

Analogicky definujeme Las Vegas zložitosnú triedu ZNC

- *algoritmus nerobí chybu*
- *plynmiálny počet procesorov*
- *očakávaný čas je polylogaritmický*

### 5.1.1 Triedenie na PRAMoch

Máme

- $n$  vstupov a  $n$  procesorov
- pre jednoduchosť predpokladáme, že sú rôzne— $a_i \neq a_j$  pre  $i \neq j$
- ZNC— smerujeme k  $O(\log n)$  algoritmu v očakávanom prípade, čo bude výsledkom celkovo  $O(n \log n)$  operácií
- uvažujeme CREW

Začnime implementáciu pravdepodobnostného QuickSortu

---

#### Algoritmus 30 PQS

---

Na začiatku má každý procesor jeden vstup;  $P_i$  je  $i$ -ty procesor

- 1: **if**  $n = 1$  **then** stop;
  - 2: zvol splitter náhodne z  $n$  prvkov
  - 3: každý procesor  $P_i$  si určí, či je jeho prvok väčší alebo menší ako splitter
  - 4: nech  $j$  je rank(poradie) splittera
  - 5: **if**  $j \notin [n/4, 3n/4]$  **then** krok je chybný a treba ho opakovať
  - 6: **else** splitter sa presunie do  $P_j$  //  $j \in [n/4, 3n/4]$ , pokračujeme ďalej
  - 7: menšie prvky sa presunú do  $P_i$ ,  $i < j$
  - 8: väčšie prvky sa presunú do  $P_i$ ,  $i > j$
  - 9: rekurzívne sa utriedi  $P_1 \dots P_{j-1}$  a  $P_{j+1} \dots P_n$
- 

Ľahko vidno, že v najlepšom prípade sa splitter vždy zvolí úspešne (teda na prvýkrát). V takom prípade by triedenie bolo skončené v čase  $O(\log n)$ , ak by sme vedeli prerozdelenie v riadkoch 6-8 spraviť v konštantnom počte krokov.

*implementácia  
prerozdelenia*

Začnime priamočiarou implementáciou. Každý procesor  $P_i$

- nastaví svoj bit na hodnotu 0/1 podľa svojho vzťahu k splitteru:

$$b_i = \begin{cases} 1, & a_i < \text{splitter} \\ 0, & a_i > \text{splitter} \end{cases}$$

- v logaritmickom čase "pomocou binárneho stromu" vypočíta hodnotu

$$s_i = \begin{cases} \sum_{t \leq i} b_t, & b_i = 1 \\ \sum_{t > i} b_t, & b_i = 0 \end{cases}$$

pomocou ktorej určí presné miesto, kam sa má zaradiť pre ďalšie rekurzívne volanie triedenia

- $a_i$  sa umiestni na pozíciu

$$\begin{cases} s_i & \text{ak } b_i = 1 \\ n - s_i + 1 & \text{ak } b_i = 0 \end{cases}$$

Popísaná metóda spravila prerozdelenie v logaritmickej počte krokov, čo spolu s logaritmickej hĺbkou rekurzívne vedie s veľkou pravdepodobnosťou k počtu krokov  $O(\log^2 n)$ .

Pri popísanom postupe sme nevyužili, že viac procesov by mohlo rekurzívne triediť viac skupín. Pokúsime sa teda vytvoriť rozumný počet rozumne veľkých "navzájom usporiadaných" množín prvkov  $S_1, \dots, S_k$ : *vylepšenie*

$$\forall x \in S_r \begin{cases} x > y, & y \in S_p, p \leq r \\ x < y, & y \in S_p, p \geq r \end{cases}$$

Pod rozumnou veľkosťou máme na mysli  $n^{1-\varepsilon}$  pre nejakú konštantu  $\varepsilon$ . Uvažujme CREW s  $n^2$  procesormi, na začiatku je vstup v  $P_1, \dots, P_n$ . Utriediť to vieme v  $O(\log n)$  krokoch

–  $P_{i,j}$  porovná  $P_i$  a  $P_j$

–  $P_i$  si vypočíta  $S_i$  a presunie sa, kam treba ◊

Majme  $n$  procesorov,  $n$  prvkov

*n procesorov, n prvkov*

- nech procesory  $P_1, \dots, P_r$  obsahujú utriedené pole prvkov v  $P_{r+1}, \dots, P_n$  sú ostatné prvky
- utriedené prvky sú splittre - pre  $1 \leq j \leq r$  označuje  $S_j$  j-ty najmenší splitter
- chceme  $n - r$  neutriedených prvkov vložiť medzi splittre tak, aby na konci
  1. každý prvok bol v jednom procesore
  2. ak  $i(S_j)$  je index procesora, v ktorom je umiestnený  $S_j$ , tak

$$\begin{cases} k < i(S_j), & P_k \text{ obsahuje prvok } < S_j \\ k > i(S_j), & P_k \text{ obsahuje prvok } > S_j \end{cases}$$

Ak máme  $n$  prvkov,  $\sqrt{n}$  utriedených splitterov, tak popísaný insert realizujeme v  $O(\log n)$  krokoch: *realizácia insertu*

–  $O(\log r)$  na zistenie intervalu, kam prvok patrí

–  $O(\log n)$  na zistenie počtu prvkov menších ako ten-ktorý prvok

–  $O(1)$  na umiestnenie prvku do procesora, kam patrí

---

### Algoritmus 31 BoxSort

---

- 1: náhodne určí  $\sqrt{n}$  prvkov z množiny  $n$  vstupov. Utriedi ich pomocou všetkých  $n$  procesorov v  $O(\log n)$  krokoch
  - 2: vlož ostatné prvky pomedzi splittre v  $O(\log n)$  krokoch
  - 3: **if** počet prvkov v jednom boxe  $> \log n$  **then** rekurzia v boxe
  - 4: **else** LogSort (triedenie  $m$  prvkov pomocou  $m$  procesorov v  $O(m)$  krokoch; napr pár-nepár)
- 

Pomocou Chernoffovho odhadu sa dá ukázať

**Fakt 5.1** Ak máme  $m$  náhodne vybraných splittrov z  $m^2$  rôznych prvkov, potom pravdepodobnosť, že box má viac ako  $bm$  prvkov, je najvyššia  $ma^b$ ,  $a < 1$ .

Pre účely analýzy času priradíme priebehu algoritmu strom:

*analýza času*

- vrchol je box

- synovia boxy, ktoré vznikli rozdelením podľa náhodne vybraných splittrov
- v listoch sú boxy veľkosti  $< \log n$

Potom čas je vážená dĺžka cesty z koreňa do listu, pričom váha hrany na ceste je čas spotrebovaný na prechod od otca k synovi. Váha hrany je zrejme logaritmus veľkosti otcovského boxu. Ukážeme, že súčet logaritmov veľkostí boxov na ceste z koreňa do listu je s veľkou pravdepodobnosťou  $O(\log n)$ .

Rekurzia začína s intervalom  $[1, n]$ , ktorý postupne zmešujeme na  $I_0, I_1, \dots$ . Zaujímá nás, aká je pravdepodobnosť, že syn otca-boxu veľkosti z  $I_k$ , bude mať tiež veľkosť z  $I_k$ .

Uvažujme nasledovné konštanty

- $\gamma : 1/2 < \gamma < 1$
- $d : 1 < d < 1/\gamma$
- $\forall k \in \mathbb{N}$  definujeme
 
$$\begin{aligned} \tau_k &= d^k \\ \rho_k &= n^{\gamma^k} \\ I_k &= [\rho_{k+1}, \rho_k] \end{aligned}$$

**Cvičenie 5.1** Ukážte, že  $\rho_k < \log n$  pre  $k \leq c \log \log n$ ,  $c = c(\gamma)$  ( $c$  je konštanta závislá od  $\gamma$ )

Na základe cvičenia 5.1 vieme, že máme  $O(\log \log n)$  intervalov  $I_k$ . Ku každému boxu  $B$  môžeme na základe veľkosti priradiť číslo intervalu, do ktorého patrí jeho veľkosť:  $\alpha(B) = k$  ak  $|B| \in I_k$ . Vzhľadom na definíciu hraníc intervalu  $I_k$  je čas na split boxu  $B$  úmerný  $O(\log(\rho_{\alpha(|B|)}))$ .

- Majme teda cestu  $\xi$  z koreňa do listu, ktorá je určená boxami, cez ktoré prechádza:  $\xi = (B_1, \dots, B_t)$ . Keďže čas algoritmu je  $O(\log n + \max_{\xi} \sum_{j=1}^t \log \rho_{\alpha(|B_j|)})$ , zaujímame sa o sumu

$$\sum_{j=1}^t \log \rho_{\alpha(B_j)}$$

- Nech  $\xi = (B_1, \dots, B_t)$ . Hovoríme, že *udalosť*  $E_{\xi}$  *platí*, ak sa v postupnosti  $\alpha(B_1), \dots, \alpha(B_t)$  žiadna hodnota  $k$  nevyskytuje viac ako  $\tau_k$ -krát; pritom  $1 \leq k \leq c \log \log n$
- Ak udalosť  $E_{\xi}$  platí, potom počet krokov PRAMu, ktoré spotrebuje na ceste  $\xi$ , je nanajvyšš

$$O\left(\log n + \sum_{k=1}^{\infty} \tau_k \gamma^k \log n\right) = O(\log n)$$

Uvedomme si:  $\tau_k = d^k$ ,  $\gamma \cdot d < 1$ ,  $\log n \sum_{k=1}^{\infty} \tau_k \gamma^k = \log n \sum_{k=1}^{\infty} (d\gamma)^k = O(\log n)$

**Lema 5.2** Existuje konštanta  $\beta > 1$  tak, že  $E_{\xi}$  platí s pravdepodobnosťou aspoň  $1 - \exp(-\log^{\beta} n)$

**Dôkaz:** Načrtne postup dôkazu

- pomocou Faktu 5.1 ohraničíme pravdepodobnosť, že  $\alpha(B_{j+1}) = \alpha(B_j)$
- spočítame pravdepodobnosť, že pre konkrétne  $k$  sa toto  $k$  vyskytuje v postupnosti  $\alpha(B_1), \dots, \alpha(B_t)$  aspoň  $\tau_k$  krát
- spočítame pravdepodobnosť, že pre  $1 \leq k \leq c \log \log n$  sa  $k$  vyskytuje v postupnosti  $\alpha(B_1), \dots, \alpha(B_t)$  aspoň  $\tau_k$  krát

□

Keďže počet ciest  $\xi$  (resp. listov) je nanajvyšš  $n$ , tak sčítaním dostaneme

**Veta 5.3** Existuje konštanta  $b > 0$  tak, že s pravdepodobnosťou aspoň  $1 - \exp(-\log^b n)$  skončí algoritmus *BoxSort* v  $O(\log n)$  krokoch.

### 5.1.2 Maximálna nezávislá množina

Majme neorientovaný graf  $G = (V, E)$  taký, že o počte hrán  $m$  platí  $m = \Omega(n)$ . Hovoríme, že podmnožina  $I \subseteq V$  je nezávislá v  $G$  ak neexistuje hrana v rámci množiny  $I$ , presnejšie  $I \times I \cap E = \emptyset$ .

- $\Gamma(v)$  označuje množinu susedov vrchola  $v$
- $d(v) = |\Gamma(v)|$
- množina je *maximálna*, ak sa nedá zväčšiť (nie je obsiahnutá vo väčšej)
- určiť veľkosť *počtom maximálnej* nezávislej pomnožiny, teda maximum  $\max$ , pre ktoré existuje maximálna množina mohutnosti  $\max$ , je NP-ťažké. Naproti tomu jednoduchý greedy algoritmus nájde maximálnu množinu ľahko (alg.MIS)

---

#### Algoritmus 32 MIS

---

```

1: $I \leftarrow \emptyset$
2: for $v = 1$ to n do
3: if $I \cap \Gamma(v) = \emptyset$ then $I \leftarrow I \cup \{v\}$

```

---

Výstupom Algoritmu 32 je lexikograficky prvá MIS, označme ju LFMIS. Ak graf bol zadaný zoznamom susedov, je zložitosť algoritmu  $O(m)$ .

Vie sa, že existencia RNC algoritmu pre LFMIS by znamenala  $P = RNC$ , čo je nepravdepodobné. Máme teda paradox – efektívny sekvenčný algoritmus je jednoduchý, ale paralelne to ide "ťažko". Keď upustíme od toho, aby sme našli LFMIS, pôjde to aj paralelne (v polylogaritmickej čase na polynomiálnom počte procesorov)

**Cvičenie 5.2** Zamyslite sa nad EREW pre overenie toho, že  $I$  je MIS pomocou  $O(m/\log m)$  procesorov v čase  $O(\log m)$ .

Jednoduchou úpravou získame pravdepodobnostnú verziu algoritmu MIS – PMIS. *randomizácia sekvenčného* Namiesto toho, aby sme sa snažili pridať najmenší prvok, budeme pridávať náhodný prvok (vrchol)  $v$ ; po jeho pridaní odstránime jeho množinu susedov  $\Gamma$ .

Pravdepodobnostná verzia sa dá sparalizovať—nájdeme nezávislú množinu vrcholov  $S$ , pridáme ju k budovanej nezávislej množine  $I$  a odstránime  $S$  spolu s  $\Gamma(S)$ . *paralelizácia* Musíme ale vyriešiť dva problémy

1. každá iterácia sa musí realizovať efektívne
2. počet iterácií by mal byť malý; to si vyžaduje "rozumnú" veľkosť  $S \cup \Gamma(S)$ . Stratégia pre voľbu bude založená na počte hrán.

Zvolíme *náhodne* dostatočne veľkú množinu  $R \subseteq V$ . Táto zrejme nebude nezávislá. Nezávislú z nej spravíme tak, že z dvojice vrcholov  $u, v \in R : (u, v) \in E$  zachováme ten, ktorý má vyšší stupeň.

Je zrejme, že Algoritmus P-MIS

- naozaj pridáva nezávislú množinu  $S$
- skončí určite po lineárnom počte iterácií

Chceli by sme, aby očakávaný počet iterácií bol  $O(\log)$ .

**Cvičenie 5.3** Zamyslite sa nad EREW implementáciou Algoritmu PMIS, ktorá pri realizácii jednej iterácie použije  $O(n + m)$  procesorov a pracuje v čase  $O(\log n)$ .

Analýzu času Algoritmu MIS spravíme na základe analýzy počtu hrán odstránených v jednej iterácii. Hovoríme, že *počet odstránených hrán*

**Algoritmus 33** P-MIS

---

```

1: $I \leftarrow \emptyset$
2: repeat
3: for all $v \in V$ do(paralelne)
4: if $d(v) = 0$ then pridaj v do I a odstráň v z V
5: else $Mark(v)$ s pravdepodobnosťou $1/2d(v)$
6: for all $(u, v) \in E$ do(paralelne)
7: if $Mark(u) \wedge Mark(v)$ then odznač vrchol s nižším stupňom
8: for all $v \in V$ do
9: if $Mark(v)$ then zaraď v do S
10: $I \leftarrow I \cup S$
11: odstráň $S \cup \Gamma(S)$ z V , incidentné hrany odstráň z E
12: until $V = \emptyset$

```

---

– vrchol  $v \in V$  je *dobrý*, ak má aspoň  $d(v)/3$  susedov stupňa  $\leq d(v)$

– hrana  $(v, u)$  je *dobrá*, ak aspoň jeden z vrcholov  $u, v$  je dobrý.

Definíciu robíme vzhľadom k existujúcej verzii grafu. Dobrý vrchol má rozumnú šancu, aby jeho sused nízkeho stupňa bol v  $S$ , a teda odstránený z  $V$ . Ukážeme, že dobrých hrán je veľa a keďže dobré vrcholy sa pravdepodobne odstránia, v každej iterácii odstránime veľa dobrých hrán.

**Lema 5.4** *Nech  $v \in V$  je dobrý vrchol stupňa  $d(v) > 0$ . Potom pravdepodobnosť, že niektorý jeho sused  $w \in \Gamma(v)$  bude označený, je aspoň  $1 - \exp(-1/6)$*

**Dôkaz:** Každý vrchol  $v$  je označený nezávisle s pravdepodobnosťou  $1/2d(v)$ . Vrchol  $v$  je dobrý, preto aspoň  $d(v)/3$  vrcholov v  $\Gamma(v)$  má stupeň nanajvyš  $d(v)$ . Pravdepodobnosť, že *žiadny* nie je označený, je nanajvyš

$$\left(1 - \frac{1}{2d(v)}\right)^{d(v)/3} = \left(1 - \frac{1}{2d(v)}\right)^{2d(v)/6} \leq e^{-1/6}$$

□

**Lema 5.5** *Počas ľubovoľnej iterácie je označený vrchol  $w$  vybratý do  $S$  s pravdepodobnosťou aspoň  $1/2$ .*

**Dôkaz:** Vrchol  $w$  odznačíme, ak má označeného suseda  $x$  stupňa aspoň  $d(w)$ ;  $d(x) \geq d(w)$ . Vieme, že vrchol  $x$  sme označili s pravdepodobnosťou nanajvyš  $1/2d(w)$ . Pritom takých  $x$  nie je viac ako  $d(w)$

$$\begin{aligned} Pr[\text{označený } w \text{ ostane v } S] &\geq 1 - Pr[\exists x \in \Gamma(w) : d(x) \geq d(w), x \text{ je označený}] \\ &\geq 1 - |\{x \in \Gamma(w) : d(x) \geq d(w)\}| \cdot \frac{1}{2d(w)} \\ &\geq 1 - \sum_{x \in \Gamma(w)} 1/2d(w) = 1 - d(w) \frac{1}{2d(w)} = 1/2 \end{aligned}$$

□

Nech

$v$  je dobrý vrchol,  $d(v) > 0$

$E$  je udalosť, že nejaký vrchol z  $\Gamma(v)$  bude označený

$w$  označený vrchol v  $\Gamma(v)$  s najmenším číslom. S pravdepodobnosťou aspoň  $1/2$  sa  $w$  dostane do  $S$ . Ak  $w \in S$ , potom  $v \in S \cup \Gamma(v)$

$w$  je označený s pravdepodobnosťou aspoň  $(1 - \exp(-1/6))$ , prežije s pravdepodobnosťou aspoň  $1/2$ , preto platí nasledujúca lema:

**Lema 5.6** *Pravdepodobnosť, že dobrý vrchol  $v$  patrí do  $S \cup \Gamma(S)$ , je aspoň  $(1 - \exp(-1/6))/2$*

Teraz už môžeme ohraničiť počet dobrých hrán v grafe.

# dobrých hrán

**Lema 5.7** *Počet dobrých hrán v grafe  $G = (E, V)$ , resp. jeho aktuálnej verzii, je aspoň  $|E|/2$*

**Dôkaz:** Zorientujeme hrany tak, aby ukazovali na vrchol vyššieho stupňa; pri rovnosti na vrchol s väčším indexom. Nech pre vrchol  $v$  označuje  $d_i(v), d_o(v)$  počet prichádzajúcich, resp. odchádzajúcich hrán. Potom pre každý zlý vrchol  $v$

$$d_o(v) - d_i(v) \geq \frac{d(v)}{3} = \frac{d_o(v) + d_i(v)}{3}$$

$S, T \subseteq V$   $E(S, T)$  množina orientovaných hrán z  $S$  do  $T$

$$e(S, T) = |E(S, T)|$$

$V_G$  je množina dobrých vrcholov

$V_B$  množina zlých vrcholov

$$V = V_G \cup V_B$$

Spočítame totálny stupeň zlých vrcholov:

$$\begin{aligned} 2e(V_B, V_B) + e(V_B, V_G) + e(V_G, V_B) &= \\ &= \sum_{v \in V_B} (d_o(v) + d_i(v)) \leq 3 \sum_{v \in V_B} (d_o(v) - d_i(v)) \\ &= 3 \sum_{v \in V_G} (d_i(v) - d_o(v)) = \\ &= 3[e(V_B, V_G) + e(V_G, V_G) - (e(V_G, V_B) + e(V_G, V_G))] \\ &= 3[e(V_B, V_G) - e(V_G, V_B)] \leq 3[e(V_B, V_G) + e(V_G, V_B)] \end{aligned}$$

$$2e(V_B, V_B) \leq 2[e(V_B, V_G) + e(V_G, V_B)]$$

To znamená, že dobrých hrán je aspoň toľko, ako zlých a je ich preto aspoň  $|E|/2$ .  $\square$

Zhrňme si, čo vieme

- konštantný zlomok hrán susedí s dobrými vrcholmi
- dobrý vrchol je eliminovaný s konštantnou pravdepodobnosťou
- očakávaný počet eliminovaných hrán v jednej iterácii je konštantný zlomok existujúcich hrán

Preto je očakávaný počet iterácií  $O(\log n)$  a platí:

**Veta 5.8**  *$P$ -MIS má takú implementáciu na EREW, ktorá beží v očakávanom čase  $O(\log^n)$  pri  $O(n + m)$  procesoroch.*

### 5.1.3 Perfect matching

V tejto časti sa budeme venovať hľadaniu nezávislej množiny hrán:

- maximálne párovanie – nedá sa zväčšiť pridaním hrany; sekvenčne sa ľahko hľadá greedy metódou
- kardinalitu maximálne párovanie; existujú polynomiálne sekvenčné riešenia. Zdá sa, že všetky rozumné paralelizácie obsahujú pravdepodobnostný prístup.

- perfect matching – obsahuje všetky vrcholy grafu

Budeme sa venovať problému úplného párovania v grafe, o ktorom vieme, že toto párovanie obsahuje. Môžeme si to dovoliť, pretože rozhodovacia verzia tohto problému je z RNC. Pravdepodobnostný algoritmus je založený na nasledujúcej vete:

**Veta 5.9 (Tutte)** *Nech  $A$  je matica, ktorá vznikla z grafu  $G = (V, E)$ :*

- každej hrane  $(v_i, v_j) \in E$  priradíme premennú  $x_{i,j}, i < j$
- hodnota matice na mieste  $i, j$  je určená podľa hrán

$$A[i, j] = \begin{cases} x_{i,j}, & (v_i, v_j) \in E, i < j \\ -x_{i,j}, & (v_i, v_j) \in E, j < i \\ 0, & (v_i, v_j) \notin E, i < j \end{cases}$$

*Potom  $G$  obsahuje úplné párovanie práve vtedy, keď determinant matice  $A, \det(A)$  nie je identicky rovný 0.*

Keďže existuje NC algoritmus na výpočet determinantu matice, poskytuje veta 5.9 návod na RNC riešenie rozhodovacej verzie existencie úplného párovania v grafe:

- konštrukcia matice  $A$
- pravdepodobnostný test na  $\det(A) \neq 0$

*definície, fakty*

Zopakujme (prípomeňme) si niektoré pojmy a tvrdenia súvisiace s maticami

**(i,j) minor**  $U^{i,j}$  je matica, ktorá z matice  $U$  vznikne odstránením  $i$ -teho riadku a  $j$ -teho stĺpca

**adjoint**  $adj(U)$  je matica, ktorá má na mieste  $(i, j)$  hodnotu v absolútnej hodnote rovnú determinantu  $(i, j)$ -minora;

$$adj(U)(i, j) = (-1)^{i+j} \det(U^{i,j})$$

$$\text{Platí } U \cdot adj(U) = \det(U)$$

**Veta 5.10** *Nech  $U$  je  $n \times n$  matica, ktorej prvky sú  $k$ -bitové čísla. Potom determinant, adjoint,  $U^{-1}$  sú z NC. Presnejšie, ak  $M(n) = O(n^{2.376})$  je počet operácií spotrebovaných pri násobení 2 matic typu  $n \times n$ , potom*

- determinant počítame v čase  $O(\log^2 n)$  pomocou  $O(n^2 M(n))$  procesorov
- existuje RNC pre výpočet inverznej matice, adjoint v čase  $O(\log^2 n)$  pomocou  $O(n^{3.5} k)$  procesorov.

Problémom pri získavaní paralelného algoritmu na hľadanie úplného párovania v grafe, o ktorom vieme, že úplné párovanie má, je to, že týchto úplných párovaní môže byť veľa. Potrebujeme nejakým spôsobom "izolovať" jedno, ktoré bude viacero procesorov hľadať spoločne. Túto izoláciu spravíme vhodným ohodnotením hrán a následne hľadaním úplného párovania minimálnej váhy. Ukážeme, že pri rozumne náhodnom váhovaní je veľká šanca, že sa izolácia podarí.

*system podmnožín*

**Definícia 5.2** *Systém podmnožín je dvojica  $(X, \mathcal{F})$ , kde  $X = \{x_1, \dots, x_m\}$  je konečná množina prvkov, tzv. univerzum a  $\mathcal{F} = \{S_1, \dots, S_k, S_i \subseteq X\}$  je trieda podmnožín. Hovoríme, že rozmer tohto systému je  $m$ -veľkosť univerza.*

*Nech  $w : X \rightarrow Z^+$  je váhová funkcia. Definujme*

$$w(S) = \sum_{x_j \in S} w(x_j)$$

Platí nasledujúca izolačná lema:

**Lema 5.11 (izolačná)** *Nech  $(X, \mathcal{F})$  je systém podmnožín rozmeru  $m$ ,  $w : X \rightarrow \{1, \dots, 2m\}$  je kladná celočíselná váhová funkcia, ktorá každému prvku z  $X$  priradí náhodnú váhu z množiny  $\{1, \dots, 2m\}$ . Potom*

$$Pr[\exists \text{ jediná množina minimálnej váhy v } \mathcal{F}] \geq 1/2$$

**Dôkaz:** Predpokladáme, že každý prvok z  $X$  sa vyskytuje aspoň v jednej z podmnožín systému  $\mathcal{F}$ . Predstavme si, že váhy jednotlivým prvkom priraďujeme postupne. Nech všetky prvky z  $X$  až na  $x_i$  už majú priradenú váhu. Označme

$W_i$  minimálnu váhu množiny, ktorá *obsahuje*  $x_i$ , pričom váha  $w(x_i) = 0$ , keďže ju nepoznáme

$\overline{W}_i$  minimálnu váhu množiny, ktorá *neobsahuje*  $x_i$

$$\alpha(i) = \overline{W}_i - W_i \quad (\text{všimnime si, že } \alpha(i) \text{ môže byť aj } < 0)$$

Ako sa prejaví priradenie váhy prvku  $x_i$ ?

- Ak by sme priradili  $x_i$  nekonečne malú váhu (resp.  $-2m^2$ ), potom by každá množina minimálnej váhy musela obsahovať prvok  $x_i$ . Analogicky, nekonečne veľká váha (resp.  $2m^2$ ) by spôsobila, že žiadna minimálna množina  $x_i$  neobsahuje.
- $w(x_i) < \alpha(i)$  spôsobí, že každá množina minimálnej váhy musí obsahovať  $x_i$ , pričom každá množina, ktorá  $x_i$  neobsahuje, má váhu aspoň  $W_i + w(x_i) < W_i + (\overline{W}_i - W_i) = \overline{W}_i$
- $w(x_i) > \alpha(i)$  spôsobí, že žiadna množina minimálnej váhy neobsahuje  $x_i$ , pričom každá množina, ktorá  $x_i$  obsahuje, má váhu aspoň  $W_i + w(x_i) > \overline{W}_i$

Ak  $w(x_i) = \alpha(i)$ , nevieme jednoznačne povedať, či  $x_i$  bude alebo nebude prvkom minimálnej množiny. V tomto prípade hovoríme, že je  $x_i$  *neisté*.

Ak však  $w(x_i) \neq \alpha(i)$ , potom je prítomnosť/nepřítomnosť  $x_i$  v minimálnej množine jednoznačne určená.

$$w(x_i) = \alpha(i)$$

$$w(x_i) \neq \alpha(i)$$

Uvedomme si, že náhodná premenná  $\alpha(i)$  je nezávislá od  $w(x_i)$ , preto

$$Pr[x_i \text{ je neisté}] \leq 1/2m$$

Neistoty jednotlivých prvkov korelujú, preto

$$Pr[\text{existuje neistý prvok v } X] \leq m \cdot \frac{1}{2m} = \frac{1}{2}$$

Znamená to, že popísané náhodné priradenie váh je s pravdepodobnosťou aspoň  $1/2$  také, že žiaden prvok nie je neistý, čo znamená, že existuje jediná množina minimálnej váhy. (Existencia dvoch množín  $S_i, S_j$  minimálnej váhy by znamenala, že existuje prvok  $y \in S_i, y \notin S_j$ , čo by znamenalo, že  $y$  je neistý.)

□

Aplikácia izolačnej lemy na problém úplného párovania je priamočiara

- $X$  je množina hrán
- $\mathcal{F}$  je množina úplných párovanií
- náhodné priradenie váh jednotlivým hranám určí jednoznačne úplné párovanie minimálnej váhy.

*paralelizácia*

Môžeme teda predpokladať, že máme graf s náhodne priradenými váhami, ktorý obsahuje jediné úplné párovanie minimálnej váhy (keďže nejednoznačné je to s pravdepodobnosťou  $\leq 1/2$ , niekoľkonásobné opakovanie chybu redukuje; resp. máme Las Vegas algoritmus, ktorý to ováhovanie nakoniec predsaden nájde.)

Nech  $A$  je Tutte matica. Vytvoríme z nej maticu  $B$  tak, že namiesto  $x_{i,j}$  píšeme  $2^{w(i,j)}$ . Potom platia nasledujúce tvrdenia:

**Lema 5.12** *Nech existuje jediné minimálne úplné párovanie váhy  $W$ . Potom  $\det(B) \neq 0$  a súčasne maximálna mocnina dvojky, ktorá delí  $\det(B)$ , je  $2^{2W}$ .*

**Lema 5.13** *Nech  $M$  je jediné minimálne úplné párovanie v  $G$  váhy  $W$ . Potom*

$$(i, j) \in M \iff \frac{\det(B^{i,j})2^{w(i,j)}}{2^{2W}} \text{ je nepárne}$$

Zhrnutím predchádzajúcich úvah a tvrdení je MC algoritmus P-Match, pre ktorý platí:

**Veta 5.14** *Ak  $G$  má úplné párovanie, potom Algoritmu P-Match ho nájde s pravdepodobnosťou aspoň  $1/2$ . Pritom čas je  $O(\log^2 n)$  a počet procesorov  $O(n^{3.5}m)$*

---

#### Algoritmus 34 P-Match

---

▷ vstupom je graf s aspoň jedným úplným párovaním

- 1: **for all**  $(i, j)$ (paralelne) **do**
  - 2:     priradiť hrane  $(i, j)$  náhodnú váhu  $w(i, j) \in \{1, \dots, 2m\}$
  - 3: vypočítaj (deterministicky) Tutte maticu  $B$
  - 4: vypočítaj  $\det(B)$
  - 5: vypočítaj  $W$  také, že  $2^{2W}$  je maximálna mocnina dvojky, ktorá delí  $\det(B)$
  - 6: vypočítaj  $\text{adj}(B) = \det(B) \cdot B^{-1}$
  - 7: **for all**  $(i, j)$ (paralelne) **do**
  - 8:     vypočítaj  $r_{i,j} = \text{adj}(B)^{i,j} \cdot 2^{w(i,j)} / 2^{2W}$
  - 9:     **if**  $r_{i,j}$  je nepárne **then** pridaj  $(i, j)$  do  $M$
- 

### 5.1.4 Problém dohody/koordinačný problém

Uvažujme distribuovaný systém, v ktorom máme  $n$  rovnakých procesorov, ktoré realizujú asynchrónny výpočet. Cieľom je, aby zvolili rovnako jeden z možných výsledkov. Táto voľba sa realizuje umiestnením špeciálnej značky \* do práve jedného z  $m$  každému prístupných read/write registrov.

*konflikty*

Konflikty prístupu do spoločne zdieľaných registrov sa riešia uzamykaním – proces, ktorému sa podarilo do registra vstúpiť, ho zamkne a ostatní zatiaľ čakajú. Potom sa opäť pokúsia o vstup do registra.

*zložitosť*

Zložitosť meriame počtom prístupov do registrov. Vie sa, že deterministické riešenie vyžaduje  $\Omega(n^{1/3})$  operácií. Ukážeme, že existuje pravdepodobnostný algoritmus, ktorý tento problém rieši pomocou  $c$  operácií s pravdepodobnosťou úspechu aspoň  $1 - 2^{-\Omega(c)}$ . Pre jednoduchosť predpokladáme  $n = m = 2$ .

*synchronizovaný výpočet*

Začnime synchronizovaným prípadom. Na začiatku procesor  $P_i$  sníma bunku  $C_i$ , potom si to vymenia. Procesory  $P_1, P_2$  používajú každú bunku vlastnej pamäte  $R_1, R_2$  a boolovskú premennú  $B_1, B_2$ .

*korektnosť*

**Algoritmus 35 S-CCP**práca procesora  $P_i$ 

- 
- 1:  $P_i$  sníma  $C_i, B_i \leftarrow 0$
  - 2: **loop**
  - 3:    $R_i \leftarrow C_i$
  - 4:   **if**  $R_i = *$  **then** halt
  - 5:   **if**  $R_i = 0, B_i = 1$  **then** zapíš \* do čítaného registra a halt
  - 6:   náhodne nastav  $B_i$  a prirad'  $C_i \leftarrow B_i$
  - 7:   prehod' si snímaný register s  $P_{1-i}$
- 

Sporom ukážeme, že sa nemôže stať, aby oba procesory písali/mali v snímanom registri symbol \*. Ak by sa to malo stať, muselo by sa to udiať súčasne, nakoľko po zápise znaku \* procesor zastane. Nech sa to (kvôli sporu) stalo v  $t$ -tej iterácii. Označme  $B_i(t), R_i(t)$  obsah príslušných premenných po priradení  $R_i \leftarrow C_i$  v iterácii  $t$ . Zrejme  $R_0(t) = B_1(t)$  a  $R_1(t) = B_0(t)$ . Pritom zápis sa deje v situácii, keď  $R_i(t) = 0, B_i(t) = 1$ , čo znamená, že  $B_{1-i}(t) = 0, R_{1-i}(t) = 1$ . V tejto situácii ale procesor  $P_{1-i}$  symbol \* nezapíše. Ukázali sme, že ak algoritmus skončí, skončí korektne.

Ak majú procesory nastavené rôzne bity, potom v nasledujúcej iterácii korektne ukončenie skončia. Pravdepodobnosť, že si vygenerujú rovnaký bit, je  $1/2$ , z čoho

$$Pr[\text{výpočet ani po } t \text{ iteráciách neskončil}] = (1/2)^t$$

S pravdepodobnosťou  $1 - O(1/2^t)$  sa vykoná  $O(t)$  práce.

"Simulovanie" synchronizácie v asynchrónnom výpočte realizujeme pridaním časových pečiatok. *asynchrónny výpočet*

**Algoritmus 36 AS-CCP**práca procesora  $P_i$ ;  $T_i, B_i$  sú lokálne premenné (TimeStamp, Bit)

- 
- 1:  $P_i$  sníma náhodný register, na konci iterácie si to vymení
  - 2: **loop**
  - 3:    $P_i$  sníma svoj register— $[t_i, R_i]$
  - 4:   **if**  $R_i = *$  **then** halt
  - 5:   **if**  $T_i < t_i$  **then**  $T_i \leftarrow t_i, B_i \leftarrow R_i$
  - 6:   **if**  $T_i > t_i$  **then** zapíš \* a halt
  - 7:   **if**  $T_i = t_i, R_i = 0, B_i = 1$  **then**
  - 8:     zapíš \* do čítaného registra a halt
  - 9:   **if** otherwise **then**
  - 10:      $T_i \leftarrow T_i + 1, t_i \leftarrow t_i + 1$
  - 11:     náhodne nastav  $B_i, R_i \leftarrow B_i$  a zapíš  $[t_i, R_i]$  do snímaného registra
  - 12:     prehod' si snímaný register s  $P_{1-i}$
- 

**Veta 5.15** Pre každé  $c > 0$  je celková práca algoritmu AS-CCP väčšia ako  $c$  s pravdepodobnosťou najvyšš  $2^{-\Omega(c)}$

**Dôkaz:** Dôkaz je analogický. Všimnime si, kedy sa píše \*.

- $P_i$  na riadku 6: hodnota  $T_i^*$  v tomto okamihu je väčšia ako hodnota  $t_i^*$  v ním snímanom registri. Nech by medzitým  $P_{1-i}$  písalo niekde inde, nech  $T_{1-i}^*, t_{1-i}^*$  sú príslušné hodnoty v tom čase. *korektnosť*

- $P_{1-i}$  chce písať \* do  $C_{1-i}$   
 $P_i$  zanechalo v  $C_{1-i}$  čas  $T_i^*$ ,  $P_{1-i}$  prišlo písať s  $t_{1-i}^* \geq T_i^*$ , pritom  $T_{1-i}^*$  nemôže byť väčšie ako  $t_i^*$ , lebo  $P_{1-i}$  prišlo z  $C_i$ , kde bolo  $t_i^*$ . Preto

$$T_{1-i}^* \leq t_i^* < T_i^* \leq t_{1-i}^*$$

Keďže  $T_{1-i}^* < t_{1-i}^*$ ,  $P_{1-i}$  vlastne realizuje riadok 5, v ktorom sa nezapisuje.

- analogicky sa pre zápis na riadku 8 vyargumentuje, že

$$T_{1-i}^* \leq t_i^* = T_i^* \leq t_{1-i}^*$$

Tu by ale mohlo nastať  $T_{1-i}^* = t_{1-i}^*$ , čo vyargumentujeme rovnako ako v synchronizovanom prípade.

čas

Celkový čas je určený maximálnou hodnotou časových pečiatok. Čas rastie v riadku 10; vtedy  $t_i = T_i$  ale nie je pravda, že  $R_i = 0, B_i = 1$  a opäť je to situácia ako v synchronizovanom prípade.  $\square$

### 5.1.5 Byzantínska dohoda

Uvažujeme systém, v ktorom sú niektoré procesory byzantínske - ich výpočet je nepredvídateľný, môžu správu meniť,.. Navrhujeme algoritmy, ktoré sú korektné, ak je počet byzantínskych procesorov najvyššie  $t$ . Zaujímá nás problém dohody. Výpočet začína v situácii, keď má každý procesor náhodne nastavený svoj bit  $b_i \in \{0, 1\}$ . Cieľom výpočtu je dosiahnuť konfiguráciu, v ktorej má každý *korektný* procesor nastavenú *rovnakú* hodnotu  $d_i$ . Navyše, ak každý procesor začal s rovnakou hodnotou  $r$ , tak všetky korektné procesory skončia výpočet s hodnotou  $r$ . Deterministický algoritmus pre túto úlohu využíva  $t + 1$  kôl. Ukážeme, že pri využití náhodnosti dosiahneme algoritmus, ktorému s veľkou pravdepodobnosťou stačí konštantný počet kôl; konštanta je nezávislá od  $t$ . Predpokladáme, že používame "globálnu mincu"-všetky procesory vidia padané hodnoty.

Pravdepodobnosť vnášame do deterministického algoritmu na mieste, kde sa rozhoduje o akceptovaní/zamietaní momentálne väčšinovej voľby. Použijeme nasledujúce označenia

$$t < n/8 \quad L = \frac{5n}{8} + 1 \quad H = \frac{3n}{4} + 1 \quad G = \frac{7n}{8}$$

---

#### Algoritmus 37 ByzGen

---

práca  $i$ -teho procesora v každom kole

- 1: nastav  $b_i$
  - 2: **loop**
  - 3:   Broadcast svojej voľby
  - 4:   prijmi hlasy ostatných
  - 5:    $maj \leftarrow$  väčšina z toho, čo si prijal plus vlastný hlas
  - 6:    $tally \leftarrow$  počet prijatých hlasov, ktoré sa rovnajú  $maj$
  - 7:   **if** padla 1 **then** hranica bude  $L$
  - 8:       hranica bude  $H$
  - 9:   **if**  $tally \geq$  hranica **then**  $b_i \leftarrow maj$   $b_i \leftarrow 0$
  - 10:   **if**  $tally \geq G$  **then**  $d_i \leftarrow maj$  permanentne
- 

- Ak všetci v jednom kole volia rovnako, všetci si permanentne nastavia túto hodnotu

- Ak by dva rôzne korektné procesory vypočítali rôzne hodnoty *maj*, *tally* nie je väčšie ako  $L, H$ , preto všetci volia 0 a následne to korektné skončí
- Hovoríme, že byzantínske procesory zmenili hranicu  $x \in \{L, H\}$ , ak sa im podarilo spôsobiť, že pre dva rôzne korektné procesory  $P_i, P_j$  platí

$$tally > x \text{ v } P_i \text{ a súčasne } tally < x \text{ v } P_j$$

Keďže rozdiel medzi hranicami  $L, H$  je  $\frac{6n}{8} + 1 - \frac{5n}{8} - 1 = \frac{n}{8} > 1$ , môžu byzantínske procesory zmeniť nanajvyšš jednu z hraníc  $L, H$ . Pravdepodobnosť toho, že sa to stane, je  $1/2$

- Na základe predchádzajúceho je očakávaný počet kôl na dosiahnutie stavu s nezmenenou hranicou 2. Keď hranica nie je zmenená, všetky korektné procesory hlasujú rovnako, potom je  $tally \geq L, H, G$  a všetky korektné si nastaví  $d_i$  rovnako.

Vyargumentovali sme nasledujúcu vetu.

**Veta 5.16** *Očakávaný počet kôl Algoritmu ByzGen je konštantný.*

### 5.1.6 Voľba šéfa

Uvažujme problém voľby šéfa – z konfigurácie, v ktorej sú si všetci rovní sa potrebujeme dostať do konfigurácie, v ktorej jeden procesor je šéf a ostatní vedľa, že nie sú šéf. Situácia s voľbou šéfa na anonymnom kruhu je nasledovná

- deterministický algoritmus neexistuje
- existuje jednoduchý pravdepodobnostný algoritmus (Itai, Rodeh): všetci si vygenerujú náhodné identifikátory a najmenší sa stane šéfom. Je to Las Vegas algoritmus s malou pravdepodobnosťou dlhých výpočtov.

Prezentujeme iný pravdepodobnostný algoritmus<sup>1</sup>. Algoritmus je synchronizovaný, predpokladáme, že procesory vedľa, kde je ľavo/pravo a ich počet  $n$ .

---

#### Algoritmus 38 $O(1)$ -Leader

---

- 1: každý procesor si vygeneruje náhodný bit  $b_i$
  - 2: každý procesor pošle svoj (a každý novoprijatý) bit doľava aj doprava; prijaté bity zbiera, kým nemá reťazec  $S_i$  dĺžky  $n$
  - 3: zozbierané reťazce sa líšia len cyklickým šifrom, preto každý procesor vie určiť lexikograficky najmenší z nich; nech  $\ell$  je ten lexikograficky najmenšie spomedzi reťazcov, ktoré vznikli cyklickým šifrom  $S_i$
  - 4: **if**  $\ell$  je jednoznačne určené ( $\exists$  jediná pozícia, na ktorej sa cyklicky nachádza) **then** leadrom sa stáva procesor  $P_i$ , ktorý zozbieral reťazec  $\ell$ ;  $S_i = \ell$
  - 5: **else** procesory postupujú do ďalšieho kola
- 

Procesory postupujú do ďalšieho kola len vtedy, ak zozbieraný reťazec je periodický. *analýza*  
Pre Algoritmus Leader s veľkou pravdepodobnosťou platí, že v procesore sa použije  $O(1)$  náhodných bitov a celkovo sa teda preniesie  $O(n^2)$  bitov.

□

Medzi týmito dvomi extrémami —  $1 \iff \log \log n$  náhodných bitov, existujú ďalšie.

---

<sup>1</sup>Článok [3], kde dôraz je na minimalizácii počtu použitých náhodných bitov

---

**Algoritmus 39**  $O(\log \log)$ -Leader

---

- 1: procesor  $P_i$  si vygeneruje náhodný reťazec  $s_i$  dĺžky  $\log \log n$
  - 2: nech  $k_i$  je číslo s binárnym zápisom  $s_i$
  - 3: procesor  $P_i$  posiela symbol 1 s periódou  $2^{k_i}$
  - 4: procesor  $P_i$  si vyrobí  $n$ -bitový reťazec  $S_i$  počas  $2^i \times n, 2^{1+i} \times n$  keď prvýkrát dostal správu od suseda; neposlaná správa sa interpretuje ako posielanie 0
  - 5: **if** jediné lexikografické minimum **then**
  - 6:     leader je zvolený
  - 7: **else** procesory s minimálnou hodnotou postupujú do ďalšieho kola
- 

V očakávanom prípade bude  $n/\log n$  procesorov s rovnakou hodnotou, čo znamená, že v prvej fáze prekomunikujeme

$$\frac{n}{\log n} \sum_{i=0}^{\log n} \frac{n}{2^i} = O(n^2/\log n)$$

$O(n^2/\log n)$  bitov. Tie vzniknuté reťazce s veľkou pravdepodobnosťou nie sú periodické, čo byz znamenalo, že máme leadra. Ak áno, medzi tými minimálnymi dovolíme šéfa algoritmom  $O(1)$ -Leader. To nás stojí ďalších  $O(n(n/\log n)) = O(n^2/\log n)$  prenesených bitov.

□

## 5.2 Formálne jazyky a automaty

V tejto časti budeme skúmať silu pravdepodobnostných výpočtov v porovnaní s determinizmom, selfverifying nedeterminizmom a nedeterminizmom. Konkrétne si budeme všimáť jednosmernú komunikačnú zložitosť, počet stavov konečných automatov a polynomiálne ohraničené orákulovské výpočty.

**determinizmus** v každej konfigurácii je nanajvyšš jedna možnosť pokračovania; zložitosť je zložitosťou nájdenia riešenia, dôkazu neexistencie riešenia; predpona  $D, d$  resp. žiadna v označeniach strojov,...

**nedeterminizmus** z konfigurácie existuje konečne veľa možností pokračovania. Pre akceptovanie vyžadujeme existenciu aspoň jedného akceptujúceho výpočtu; v tomto prípade odpoveď áno v konkrétnom výpočte nemá výpovednú hodnotu. Pre zamietanie vyžadujeme zamietanie každého potenciálneho výpočtu. Zložitosť je zložitosťou overenia, že uhádnuté riešenie je korektným riešením. Používame predponu  $N, n$

**selfverifying nedeterminizmus** pribúda odpoveď neviem. Vyžadujeme  $x \in L$  potom aspoň jedna odpoveď je áno, ostatné áno alebo neviem  
 $x \notin L$  potom aspoň jedna odpoveď je nie, ostatné nie alebo neviem  
 Odpoveď áno má rovnakú váhu ako nie – platí. Zložitosť v tomto prípade je maximum zo zložitosti overenia uhádnutého kandidáta a zložitosti neexistencie riešenia. Používame predponu  $SV, sv, SVN, svn$

### 5.2.1 Jednosmerná komunikačná zložitosť

Komunikačná zložitosť je o.i. nástrojom pre získavanie dolných odhadov do VLSI. Čo to je? Majme Boolovskú funkciu  $f$   $2n$  premenných,  $f(X, Y) = f(x_1, \dots, x_n, y_1, \dots, y_n)$ , ktorej hodnotu majú spoločne počítať dva počítače  $C_I, C_{II}$ . Prvú polovicu vstupov  $X$  dostane  $C_I$ , druhú polovicu vstupov  $Y$  dostane  $C_{II}$ . Výpočet prebieha nasledovne

1.  $C_I$  spracuje  $X$
2.  $C_I$  pošle na základe komunikačného protokolu správu  $m(X)$  počítaču  $C_{II}$
3.  $C_{II}$  spracuje  $m(X), Y$  a odpovie.

Zaujímá nás dĺžka prekomunikovanej správy  $m(X)$  maximalizovaná cez všetky vstupy dĺžky  $2n$  a minimalizovaná cez všetky protokoly, ktoré počítajú  $f$ .

- $cc_1(\mathbf{P})$  označuje dĺžku najdlhšej správy, ktorú pošle  $C_I$  podľa konkrétneho protokolu  $P$
- $cc_1(\mathbf{f})$  označuje dĺžku najdlhšej správy, ktorú pošle  $C_I$  pri deterministickom výpočte Boolovskej funkcie  $f$
- $ncc_1(\mathbf{f})$  označuje dĺžku najdlhšej správy, ktorú pošle  $C_I$  pri nedeterministickom výpočte Boolovskej funkcie  $f$
- $svncc_1(\mathbf{f})$  označuje dĺžku najdlhšej správy, ktorú pošle  $C_I$  pri selfverifying nedeterministickom výpočte Boolovskej funkcie  $f$
- $lvcc_1(\mathbf{f})$  označuje dĺžku najdlhšej správy, ktorú pošle  $C_I$  pri Las Vegas výpočte Boolovskej funkcie  $f$

Hlavný výsledok, ktorý chceme ukázať, je  $lvcc_1(f) \geq cc_1(f)/2$ . Na konkrétnom jazyku ukážeme, že tento odhad sa nedá zlepšiť.

- Predpokladáme *verejný zdroj náhodných bitov* (public random source) - padanie mince vidia všetci

- Komunikačnú zložitosť definovanú pre Boolovské funkcie "prenesieme" na jazyky pomocou homomorfizmu: k jazyku  $L$  zadefinujeme Boolovskú funkciu

$$h_n(L)[\alpha] = 1 \Leftrightarrow \alpha \in L \cap \{0, 1\}^n$$

V dôkazoch využijeme reprezentáciu Boolovskej funkcie tzv. *komunikačnou maticou*.

komunikačná  
matica

**Definícia 5.3** Komunikačná matica boolovskej funkcie  $f$   $2n$  premenných je Boolovská matica  $M(f)$  typu  $2^n \times 2^n$  taká, že

$$M(f)[u, v] = f(w)$$

**Fakt 5.17** Počet rôznych riadkov komunikačnej matice  $M(f)[u, v]$  sa rovná počtu rôznych správ, ktoré treba pri spracovaní uv poslať.

**Dôkaz:** Ak pošle počítač  $C_I$  identifikáciu riadku  $M(f)[X]$ , počítač  $C_{II}$  môže spolu so svojím vstupom  $Y$  vypočítať hodnotu  $f(XY)$ . Preto je počet rôznych riadkov nanaajvyš taký veľký ako počet rôznych správ.

Naopak — nech by počet rôznych správ bol menší ako počet rôznych riadkov. To by znamenalo, že pre dva rôzne vstupy  $X_1, X_2$  do  $C_I$  je poslaná správa rovnaká, a teda  $f(X_1Y) = f(X_2Y)$  pre všetky vstupy  $Y$  do  $C_{II}$ . Avšak príslušné riadky boli rôzne —  $M(f)[X_1] \neq M(f)[X_2]$ . To znamená, že existuje taký vstup  $Y'$ , že  $f(X_1Y') \neq f(X_2Y')$   $\square$

Pri dôkaze hlavného tvrdenia využijeme jazyk

definícia jazyka

$$L = \{XY \in \{0, 1\}^* \mid |X| = |Y| \ \& \ (Y = 0^i 1z) \Rightarrow (X_{i+1} = 1)\}$$

**Lema 5.18** Pre vyššie definovaný jazyk  $L$  platí

1.  $cc_1(h_{4n}(L)) = 2n$
2.  $lvcc_1(h_{4n}(L)) = n$
3.  $svncc_1(h_{4n}(L)) \leq \lceil \log_2 2n \rceil + 1$

**Dôkaz:**

1.  $cc_1(\mathbf{h}_{4n}(\mathbf{L})) = 2\mathbf{n}$

Na základe faktu 5.17 dostávame, že jednosmerná komunikačná zložitosť je rovná logaritmu počtu rôznych riadkov komunikačnej matice. Súčasne ľahko vidno, že pre jazyk  $L$  sú riadky komunikačnej matice navzájom rôzne. Ak by tomu tak nebolo, existovali by dva *rôzne* vstupy  $U$  a  $V$  do  $C_I$ , pre ktoré by odpovedajúce riadky boli rovnaké:  $M(f)[U] = M(f)[V]$ . Keďže  $U \neq V$  tak existuje index  $i + 1$  tak, že  $U_{i+1} \neq V_{i+1}$ , napr.  $U_{i+1} = 1$ . Potom  $1 = M(f)[U, 0^i z] = M(f)[V, 0^i z] = 0$ .

2.  $lvcc_1(\mathbf{h}_{4n}(\mathbf{L})) = \mathbf{n}$

Randomizáciu využijeme na uhádnutie pozície prvej 1 v slove  $Y$ . Počítač  $C_I$  sa náhodne (podľa výsledku verejnej mince) rozhodne, či je prvá jednotka vstupu  $C_{II}$  v prvej alebo druhej polovici a podľa toho pošle príslušnú polovicu svojho vstupu. V prípade, že  $C_I$  poslal nesprávnu polovicu vstupu, počítač  $C_{II}$  povie, že nevie, inak odpovie korektne podľa správy, ktorú dostal. S pravdepodobnosťou  $1/2$  sa trafíme správne a teda odpoveď "neviem" bude nanaajvyš na polovici správ.

3.  $svncc_1(\mathbf{h}_{4n}(\mathbf{L})) \leq \lceil \log_2 2\mathbf{n} \rceil + 1$

Nech  $X$  je vstup do  $C_I$ . Jeden z možných komunikačných protokolov je takýto:

- $C_I$  uhádne  $i$  a pošle správu  $m(X) = i, X_{i+1}$
- $C_{II}$  overí, že prvá jednotka je na pozícii  $i + 1$  a akceptuje/zamieta podľa toho, či je  $X_{i+1}$  rovné 1 alebo 0.

Odpoveď  $C_{II}$  je korektná, ak  $i$  bolo uhádnuté správne, inak hovorí "neviem"

□

A teraz sme už pripravení na avizované tvrdenie.

**Veta 5.19**  $lvcc_1(f) \geq cc_1(f)/2$

**Dôkaz:** Začneme dolným odhadom, ktorý vyplýva z lemy 5.18; pre jazyk

dolný odhad

$$L = \{XY \in \{0, 1\}^* \mid |X| = |Y| \& (Y = 0^i 1z) \Rightarrow (X_{i+1} = 1)\}$$

je Las Vegas jednosmerná komunikačná zložitosť rovná polovici deterministickej jednosmernej komunikačnej zložitosti.

Keďže je  $cc_1(f)$  logaritmus počtu rôznych riadkov komunikačnej matice, pokúsime sa vyargumentovať, že v Las Vegas komunikačnej matici, ktorá obsahuje nielen symboly 0,1 ale aj 2 – neviem, musí existovať dostatočne veľa rôznych riadkov.

horný odhad

Uvedomme si, že Las Vegas algoritmus s verejnou mincou je vlastne množina deterministických protokolov  $P_1, \dots, P_m$ , ktoré zodpovedajú príslušným postupnostiam náhodných rozhodovaní. Každému protokolu teraz môžeme priradiť komunikačnú maticu, pričom  $M(P_i)[U, V] \in \{0, 1, 2\}$ . Keďže ide o Las Vegas výpočet, pravdepodobnosť výskytu 2 v týchto maticiach je nanajvyš 1/2

$$\frac{\sum_{i=1, UV \in \{0,1\}^*}^m M(P_i)[U, V] = 2}{m2^{4n}} \leq \frac{1}{2}$$

Cieľom je vyargumentovať, že medzi týmito deterministickými protokolmi existuje taký, ktorý má aspoň  $\sqrt{r(M(f))}$  rôznych riadkov; tu  $r(M(f))$  označuje počet rôznych riadkov matice  $M(f)$ .

Pomôžeme si váženou verziou komunikačnej matice:

$$M := M(f)$$

$r$  je počet rôznych riadkov v matici  $M$

$c$  je počet rôznych stĺpcov v matici  $M$

$w : \{1, \dots, r\} \times \{1, \dots, c\} \rightarrow \mathbb{R}$  je váhová funkcia.

Váhovaciú funkciu  $w$  definujeme iteratívne. Nech  $I$  "indexuje" množinu nezávislých riadkov

definícia  $w$ 

Začíname s prvým stĺpcom:  $i = 1, I = \{1, \dots, r\}$ . Rozlišujeme dva prípady:

 $i = 1$ 

– ak je prvý stĺpec monochromatický, tak  $w(j, 1) = 0 \quad \forall j$

– ak prvý stĺpec nie je monochromatický, potom nech  $a|I|$  spomedzi riadkov v  $I$  má v prvom stĺpci hodnotu 0,  $(1 - a)|I|$  tam má 1. Potom váhu priradíme nasledovne

$$w(j, 1) = \begin{cases} \log_2(1/a), & M[j, 1] = 0 \\ \log_2(1/(1 - a)), & \text{inak} \end{cases}$$

Nech  $(i - 1)$ ý stĺpec nie je monochromatický. Spravíme rozklad  $I$  na dve podmnožiny  $I_0, I_1$  podľa hodnoty v stĺpci  $(i - 1)$ . Prvky v  $i$  tom stĺpci váhujeme analogicky ako pre prípad  $i = 1$ , ale zvlášť pre  $I_0$  a zvlášť pre  $I_1$ .

 $i > 1$

Čo vieme povedať o takto ováhaných maticiach? Nech

$$R \subseteq \{1, \dots, r\}$$

$\text{diff}(R) = \{j \mid \exists i_1, i_2 \in R : M[i_1, j] \neq M[i_2, j]\}$ ; toto je zrejme množina stĺpcov, ktoré vzhľadom na riadky z  $R$  nie sú monochromatické

**Lema 5.20** *Pri vyššie použitých označeniach platí*

1.  $\forall (i, j) \in \{1, \dots, r\} \times \{1, \dots, c\} \quad w(i, j) \geq 0$
2.  $\forall i \in \{1, \dots, r\} \quad \sum_{j=1}^c w(i, j) = \log_2 r$
3.  $\forall R \subseteq \{1, \dots, r\} \quad \sum_{j \in \text{diff}(R)} \sum_{i \in R} w(i, j) \geq |R| \cdot \log_2 |R|$

**Dôkaz:**

1.  $w(i, j) \geq 0$  vyplýva priamo z konštrukcie
2. Budeme postupovať indukciou vzhľadom na počet riadkov  $r$ .

$r = 1$

Každý stĺpec je monochromatický, preto  $w(i, j) = 0 = \log_2 1$

$r > 1$

Nech stĺpec 1 nie je monochromatický; spravíme rozklad  $I$  na  $I_0, I_1$ , pričom  $|I_0| = a|I| = ar$ . Na  $I_0, I_1$  môžeme použiť indukčný predpoklad; pre  $I_0$  teda dostávame

$$\begin{aligned} \forall i \in I_0 \quad \sum_{j=2}^a w(i, j) &= \log ar \\ &+ \\ \underbrace{w(i, 1) = \log_2(1/a)}_{\log_2 ar + \log_2(1/a) = \log_2 r} & \qquad // M(i, 1) = 0 \end{aligned}$$

Analogicky pre  $I_1$ .

3. Opäť dôkaz indukciou.
- $|R| = 1 \quad 0 \geq 1 \cdot \log_2 1 = 0$   
 $|R| > 1$

Nech prvý stĺpec nie je monochromatický. Spravíme rozklad  $R$  na  $R_0, R_1$  a potom použijeme indukčný predpoklad. Využijeme nasledujúci fakt

$$\begin{aligned} \sum_{j \in \text{diff}(R)} \sum_{i \in R} w(i, j) &= \\ &= \sum_{\substack{i \in R \\ j=1}} w(i, j) + \sum_{\substack{j \in \text{diff}(R) \\ j \neq 1}} \sum_{i \in R} w(i, j) \\ &\geq \underbrace{|R_0| \log_2(1/a) + |R_1| \log_2(1/(1-a))}_{\log_2 \frac{|R_0|}{a} + \log_2 \frac{|R_1|}{1-a}} + \underbrace{|R_0| \log_2 |R_0| + |R_1| \log_2 |R_1|}_{\geq (|R_0| + |R_1|) \log_2 (|R_0| + |R_1|)} \\ &= |R_0| \log_2 \frac{|R_0|}{a} + |R_1| \log_2 \frac{|R_1|}{1-a} \\ &\geq (|R_0| + |R_1|) \log_2 (|R_0| + |R_1|) \qquad // \text{využívame Fakt 5.21} \end{aligned}$$

◇*lema 5.20*

Máme Las Vegas protokol  $P'$  pre funkciu  $f$ , ktorá je reprezentovaná komunikačnou maticou  $M(f)$ , maticu máme ováhanú váhovacou funkciou  $w$ .

(\*) Existuje teda deterministický protokol  $P \in \{P_1, \dots, P_m\}$ , pre ktorý súčet  $S(P)$  váh prvkov matice  $M(P)$  s hodnotou 2 je nanajvyš polovica celkovej váhy<sup>2</sup>

$$S(P) \leq \frac{1}{2} \sum_{i=1}^r \sum_{j=1}^c w(i, j) = \frac{r}{2} \log_2 r$$

<sup>2</sup>nezabúdajme, že výstup 2 dostávame s pravdepodobnosťou nanajvyš 1/2

Deterministický protokol rozdelí — po vhodnom pridaní dvojiek — množinu riadkov  $M(f)$  do tried  $R_1, \dots, R_k$ . Pre každú takú množinu riadkov  $R_s$  platí

$$\sum_{j \in \text{diff}(R_s)} \sum_{i \in R_s} w(i, j) \geq |R_s| \log_2 |R_s|$$

Keďže pôvodne (bez dvojiek) boli riadky z množiny  $R$  navzájom rôzne a teraz padli do jednej triedy, museli sme všetky prvky v stĺpcoch z  $\text{diff}(R_s)$  prepísať na 2. Preto  $\sum_{j \in \text{diff}(R_s)} \sum_{i \in R_s} w(i, j)$  poskytuje dolný odhad na súčet váh prvkov z  $M(R_s)$ , ktoré majú hodnotu 2.

Spojením faktu, že pre  $x = \sum x_i$

$$\sum x_i \log x_i \geq x \log x - x \log k$$

s podmienkou (\*) dostávame

$$r \log_2 k \geq \frac{r}{2} \log_2 r \geq r \log_2 r - r \log_2 k$$

z čoho  $r \leq k^2$ , resp.  $k \geq \sqrt{r}$

Vyargumentovali sme teda existenciu protokolu  $P$  odpovedajúceho konkrétnej postupnosti výsledkov padania náhodnej mince, ktorý má aspoň  $\sqrt{r}$  rôznych riadkov, resp. posiela aspoň  $\sqrt{r}$  rôznych správ.  $\square$

**Fakt 5.21**  $\forall x, y \geq 0 \ a \in [0, 1] \quad x \cdot \log_2 \frac{x}{a} + y \cdot \log_2 \frac{y}{1-a} \geq (x+y) \log_2 (x+y)$

**Dôkaz:** Pri dôkaze vychádzame z ekvivalencie

$$x \log \frac{x}{a} + y \log \frac{y}{1-a} \geq (x+y) \log(x+y) \Leftrightarrow \frac{x}{x+y} \log \frac{x}{a} + \frac{y}{x+y} \log \frac{y}{1-a} \geq \log(x+y)$$

$$\begin{aligned} \frac{x}{x+y} \log \frac{x}{a} + \frac{y}{x+y} \log \frac{y}{1-a} &= \underbrace{\frac{x}{x+y} \log \frac{x}{x+y} \frac{x+y}{a}}_A + \underbrace{\frac{y}{x+y} \log \frac{y}{x+y} \frac{x+y}{1-a}}_B \\ A &= \frac{x}{x+y} \log \frac{x}{x+y} \frac{1}{a} + \frac{x}{x+y} \log(x+y) \\ B &= \frac{y}{x+y} \log \frac{y}{x+y} \frac{1}{1-a} + \frac{y}{x+y} \log(x+y) \\ C &= \frac{x}{x+y} \log \frac{x}{x+y} \frac{1}{a} + \frac{y}{x+y} \log \frac{y}{x+y} \frac{1}{1-a} \geq 0^3 \\ A+B &= C + (x+y) \log(x+y) \geq \log(x+y) \end{aligned}$$

$\square$

### 5.2.2 Konečné automaty

Budeme sledovať vplyv determinizmu/nedeterminizmu/selfverifying nedeterminizmu/Las Vegas na počet stavov minimálneho konečného automatu; označenie postupne  $s(L), ns(L), svns(L), lvs(L)$ . Klasický výsledok hovorí, že  $s(L) \sim 2^{ns(L)}$ . Dolný odhad sa dokazuje pomocou jazyka

$$L_k = \{w \in \{0, 1\}^* \mid w = u1v, |v| = k-1\}$$

Začnime jednoduchým pozorovaním.

**Fakt 5.22**  $\max\{ns(L), ns(L^C)\} \leq svns(L) \leq 1 + ns(L) + ns(L^C)$

**Dôkaz:** Prvú nerovnosť dokážeme transformáciou selfverifying konečného automatu *SVNFA* na nedeterministické konečné automaty rozpoznávajúce  $L$ , resp.  $L^C$ .

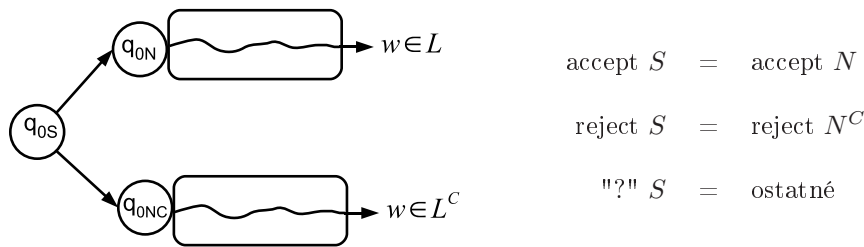
Majme *SVNFA* rozpoznávajúci jazyk  $L$ . Nedeterministický konečný automat *NFA* rozpoznávajúci jazyk  $L$  získame zaradením stavov "neviem" medzi zamietajúce stavy.  $ns(L) \leq svns(L)$

$ns(L^C) \leq svns(L)$  Pri konštrukcii *NFA* <sup>$C$</sup>  pre rozpoznávanie jazyka  $L^C$  si stačí uvedomiť, že

$$w \in L^C \Leftrightarrow \text{neexistuje akceptujúci výpočet SVNFA na slove } w$$

Automat *NFA* <sup>$C$</sup>  získame z automatu *SVNFA* zaradením zamietajúcich stavov *SVNFA* medzi akceptujúce *NFA* <sup>$C$</sup> , a stavov "neviem" medzi zamietajúce.

$svns(L) \leq 1 + ns(L) + ns(L^C)$  Nech  $N$  a  $N^C$  sú *NFA* rozpoznávajúce  $L, L^C$ . Ich jednoduchým spojením získame selfverifying automat  $S$  pre rozpoznávanie  $L$ . Automat sa rozhodne, či slovo patrí alebo nepatrí do jazyka a na *potvrdenie* svojho hádania použije príslušný *NFA*.



□

Vzťah počtu stavov v závislosti od modelu charakterizuje pre jazyk  $L_k$  nasledujúca veta.

**Veta 5.23** *Nech  $k \in \mathbb{N}$ . Potom*

1.  $s(L_k) = 2^k$
2.  $lvs(L_k) \leq 4 \cdot 2^{k/2} = O(\sqrt{s(L_k)})$
3.  $svns(L_k) \leq 2k + 3$
4.  $ns(L_k) = k + 1 = ns(L_k^C)$

1,3,4

**Dôkaz:** Dôkazy tvrdení 1 a 4 poznáme z FoJa. Časť 3 je dôsledok Faktu 5.22.

2

Idea konštrukcie Las Vegas konečného automatu je jednoduchá/elegantná - vyjadríme  $L_k$  ako zjednotenie disjunktných jazykov, pre ktoré máme jednoduchšie DFA. LVFA potom na začiatku uhádne, ktorá možnosť nastala a tú pôjde deterministicky overovať.

$$L_k^{odd} = \{w \in \{0, 1\}^* \mid w = u1v, |v| = k - 1 \text{ \& } |u| \text{ je nepárna}\}$$

$$L_k^{even} = \{w \in \{0, 1\}^* \mid w = u1v, |v| = k - 1 \text{ \& } |u| \text{ je párna}\}$$

Deterministické konečné automaty pre rozpoznávanie  $L_k^{odd}, L_k^{even}$  majú  $2^{k/2+1}$  stavov - stačí si pamätať hodnoty na každej druhej pozícii v suffixe dĺžky  $k$  a počítať pár/nepár.

□

Na oddelenie selfverifying nedeterminizmu a nedeterminizmu použijeme jazyk  $U_m$

$$U_m = \{u0v1w, u1v0w \mid |v| = m - 1, uvw \in \{0, 1\}^*\}$$

**Fakt 5.24** *Nech  $m \in \mathbb{N}$ . Potom*

1.  $ns(U_m) \leq 2m + 2$
2.  $ns(U_M^C) \geq 2^m$
3.  $svns(U_m) \geq 2^m$

**Dôkaz:**

1. Chceme kontrolovať, že existuje taká pozícia  $i$ , že vo vzdialenosti  $m$  od nej sa nachádza "opačný" symbol. Uhádneme pozíciu, zapamätáme symbol a  $ns(U_m) \leq 2m + 2$  počítame do  $m$ .
2. Stačí si uvedomiť, že  $L_{xx} = \{xx \mid x \in \{0, 1\}^m\} = U_M^C \cap \{0, 1\}^{2m}$ . Na rozpoznávanie jazyka  $L_{xx}$  potrebujeme  $2^m$  stavov. Ak by totiž NFA nemal  $2^m$  stavov,  $ns(U_M^C) \geq 2^m$  existovali by rôzne slová  $x, y$  dĺžky  $m$ , po prečítaní ktorých by bol automat v akceptujúcich výpočtoch na slovách  $xx, yy$  v tom istom stave. Potom by ale existoval aj akceptujúci výpočet na slove  $xy$ .
3. Vyplyva z Faktu 5.22  $svns(U_m) \geq 2^m$

□

Teraz sa vráťme k oddeleniu Las Vegas a determinizmu. Zamyslime sa nad súvisom počtu stavov konečného automatu a počtom správ jednosmerného protokolu. Ak by sme poznali príslušný minimálny konečný automat, stačí v komunikačnom protokole poslať stav, v ktorom automat skončil po dočítaní prvej časti slova. Preto

$$cc_1(h_n(L)) \leq \lceil \log_2 s(L) \rceil$$

Malý problém pri využití komunikačnej zložitosti je ten, že je definovaná ako neuniformná, zatiaľ čo konečné automaty sú uniformným výpočtovým modelom. Správime preto uniformnú verziu jednosmernej komunikačnej zložitosti.

**Definícia 5.4** *Nech  $\Sigma$  je (konečná) abeceda,  $L \subseteq \Sigma^*$ . Jednosmerný uniformný protokol nad  $\Sigma$  je dvojica  $D = (\Phi, \Psi)$ :*

$$\Phi : \Sigma^* \rightarrow \{0, 1\}^* \text{ spĺňa prefix-freeness property}$$

$$\Psi : \{0, 1\}^* \times \{0, 1\}^* \rightarrow \{\text{accept}, \text{reject}\}$$

$D = (\Phi, \Psi)$  akceptuje jazyk  $L$ ,  $L = L(D)$ , ak  $\forall x, y \in \Sigma^*$

$$\Psi(y, \Phi(x)) = \text{accept} \Leftrightarrow xy \in L$$

Zložitosť meraná počtom správ

$$mc(D) = |\{\Phi(x) \mid x \in \Sigma^*\}|$$

$$mc(L) = \min\{mc(D) \mid D \text{ akceptuje jazyk } L\}$$

K jazyku  $L$  môžeme priradiť komunikačnú maticu analogicky ako k Boolovskej funkcii. Rozdiel je v tom, že táto matica je nekonečná

$$\forall u, v \in \Sigma^* \quad M_L(u, v) = 1 \Leftrightarrow uv \in L$$

Označme  $row(L)$  počet rôznych riadkov matice  $M_L$ . Potom platí

**Fakt 5.25**  $s(L) = mc(L) = row(L)$

**Dôkaz:** Rovnosť  $s(L) = row(L)$  je dôsledkom Myhill-Nerodovej vety.

$$s(L) = row(L)$$

Súvis medzi počtom stavov a počtom správ je očividný—na jednej strane stačí ako správu poslať stav, v ktorom automat skončil po prečítaní prvej časti slova, na strane druhej môže byť stavom automatu správa, ktorú by protokol poslal, keby doteraz prečítaná časť slova bola vstupom do počítača  $C_I$ .

$$s(L) = mc(L)$$

□

**Veta 5.26** *Pre každý regulárny jazyk  $L$  platí*

$$lvs(L) \geq \sqrt{s(L)}$$

**Dôkaz:** Využijeme Vetu 5.19. Komunikačná matica regulárneho jazyka má konečne veľa rôznych riadkov. Vezmime podmaticu  $M$  matice  $M_L$ , ktorá má  $row(L) = s(L)$  rôznych riadkov. Každý LasVegas protokol počítajúci  $M$  použije aspoň  $\sqrt{row(L)} = \sqrt{s(L)}$  rôznych správ. Počítať celú maticu  $M_L$  nemôže byť ľahšie.

Ku každému LasVegas konečnému automatu vieme spraviť LasVegas uniformný protokol, ktorý pošle *presne*  $s(L)$  rôznych správ. Preto  $lvs(L) \geq \sqrt{s(L)}$

□

## Kapitola 6

# Logické obvody

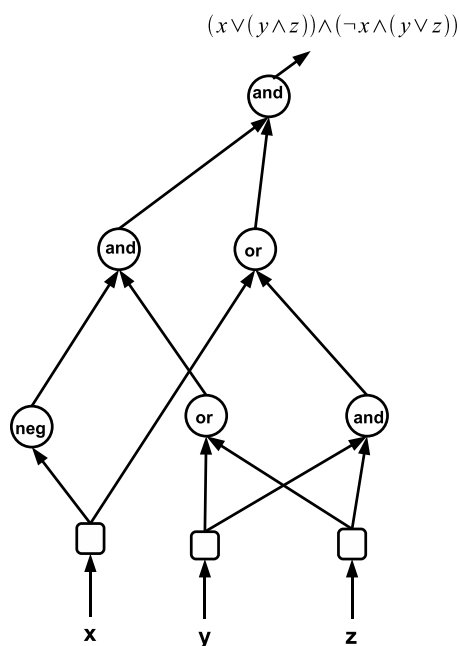
Pre naše účely je logický obvod orientovaný acyklický graf, pričom

**vstup** vstupné slovo sa do obvodu dostáva cez "listy"—špeciálne vstupné vrcholy

**vnútorné vrcholy** každý vnútorný vrchol počíta jednu z funkcií  $\neg, \wedge, \vee$ ; hodnoty porichádzajúce po vstupných hranách sú vstupné parametre funkcie, po výstupnej hrane odchádza výsledok

**výstup** výsledok výpočtu odchádza zo špeciálnych výstupných vrcholov

Z popisu je zrejmé, že obvod sa konštruje pre daný počet vstupov. Ide teda o neuniformný výpočtový model. Problém potom rieši *trieda obvodov*  $C = \{C_i\}$ , pričom  $C_i$  je obvod pre vstupy veľkosti  $i$ .



Čo sa týka zložitosti obvodu  $C$ , zvyknú sa uvažovať dve miery zložitosti

**size** veľkosťou obvodu myslíme počet vrcholov

**depth** hĺbka obvodu je maximálna vzdialenosť medzi vstupným a výstupným vrcholom; je zrejmé, že odpovedá času výpočtu obvodu; ako uvidíme neskôr, to zas odpovedá priestoru, ktorú používa sekvenčný TS na simuláciu.

Hovoríme, že  $L \subseteq \{0, 1\}^*$  má obvod polynomiálnej veľkosti, ak existuje trieda obvodov  $C = C_0, C_1, \dots$  a polynóm  $p(n)$  tak, že

- $size(C_n) \leq p(n)$
- $x \in L \Leftrightarrow C_{|x|}(x) = 1$

**Príklad 6.1** Uvažujme problém *dosiachnutelnosti*—pre vstupný graf zadaný maticou popis susednosti nás zaujíma, či existuje cesta z vrchola 1 do vrchola  $n$ . Obvod vypočíta tranzitívny uzáver matice a vráti požadovanú hodnotu.

Pre  $1 \leq i, j \leq n$  a  $0 \leq l \leq n$  máme vrchol

$$\begin{aligned} g(i, j, k) = \text{true} &\iff i \rightsquigarrow j \text{ a hodnota vnútorného vrchola } \max k \\ h(i, j, k) = \text{true} &\iff g(i, j, k) = \text{true} \text{ a prechádza aj cez vrchol } k \end{aligned}$$

Zrejme

$$\begin{aligned} h(i, j, k) &= g(i, k, k-1) \wedge g(k, j, k-1) \\ g(i, j, k) &= g(i, j, k-1) \wedge h(k, j, k) \\ \text{vstup : } &g(1, n, n) \end{aligned}$$

veľkosť

Popísaný obvod  $C$  má  $n^2$  vstupov (idú do  $g(i, j, 0)$ ),  $n^3$  vrcholov typu  $g$  aj  $h$ . Preto  $\text{size}(C) = O(n^3)$ .

**Veta 6.1** *Nech  $L \in P$ . Potom  $L$  má obvod polynomiálnej veľkosti*

**Dôkaz:** Dôkaz je konštruktívny, spravíme simuláciu Turingovho stroja obvodom. Nech TS má časovú zložitosť  $t(n)$  a priestorovú  $s(n)$ , kde  $s(n), t(n)$  sú polynómy,  $s(n) \leq t(n)$ . Nech výpočet TS na vstupe  $x$  je postupnosť konfigurácií  $C_0, C_1, \dots, C_{t(n)}$ . Potom simulujúci obvod bude mať  $t(n) + 1$  vrstiev, pričom  $i$ -ta vrstva bude reprezentovať  $C_i$ .  $\square$

Neuniformita obvodov im, v porovnaní s TS, pridáva na sile.

**Veta 6.2** *Existujú nerozhodnuteľné jazyky, ktoré majú obvody polynomiálnej veľkosti.*

**Dôkaz:** Idea dôkazu je veľmi jednoduchá. Vezmime ľubovoľný nerozhodnuteľný jazyk  $L$  nad binárnou abecedou  $\Sigma = \{0, 1\}$  a spravme k nemu unárnu verziu

$$U = \{1^n \mid \text{binárny zapis } n \in L\}$$

Je zrejmé, že jazyk  $U$  je nerozhodnuteľný, napriek tomu má obvod polynomiálnej veľkosti  $\square$

Východiskom z tejto nepeknej situácie sú tzv. uniformné obvody.

uniformné obvody

**Definícia 6.1** *Problém má uniformný obvod, ak existuje deterministický LOGSPACE ohraničený TS, ktorý pre vstup  $1^n$  vypočíta kód  $C_n$ ; keďže obvod je graf, kódom rozumieme zoznam vrcholov a zoznam hrán.*

Z popisu konštrukcie obvodu v príklade 6.1 vidno, že problém rozhodnuteľnosti je príkladom problému, ktorý má uniformný obvod.

**Veta 6.3**  *$L$  má uniformný polynomiálny obvod práve vtedy ak  $L \in P$*

**Dôkaz:** Ľahko vidno, že ak  $L \in P$ , tak v dôkaze vety 6.1 konštruovaný obvod je uniformný.



Nech  $L$  má uniformný polynomiálny obvod  $C = \{C_i\}$ ,  $M$  nech je ten LOGSPACE-stroj, ktorý ten obvod konštruuje.

- LOGSPACE stroj spraví zmysluplne (bez zacyklenia) polynomiálne veľa krokov
- simulácia obvodu je polynomiálna od jeho veľkosti, preto simulácia obvodu polynomiálnej veľkosti sa robí v polynomiálnom čase

$\square$

**Hypotéza 1** *NP úplné problémy nemajú uniformné obvody polynomiálnej veľkosti.*

Dôkaz hypotézy 1 je ekvivalentný dôkazu  $P \neq NP$ ;  $NP \in P$  totiž implikuje  $P = NP$ . Stretávame sa skôr so snahou o dôkaz ešte silnejšieho tvrdenia:

**Hypotéza 2** *NP úplné problémy nemajú obvody polynomiálnej veľkosti.*

# Literatúra

- [1] Juraj Hromkovič: *Design and Analysis of randomized Algorithms*. Springer 2005. ISBN 3-540-23949-9
- [2] Rajeev Motwani and Prabhakar Raghavan: *Randomized Algorithms*. Cambridge University Press 1995. ISBN 0 521 47465 5
- [3] B. Codenotti, P. Gemmel, P. Pudlak and J. Simon: *On the Amount of Randomness Needed in Distributed Computations* OPODIS 1997, 237-248
- [4] Pavol Ďuriš, Juraj Hromkovič, José D.P.Rolim and Georg Schnitger: *On the Power of Las vegas for One-way Communication Complexity, Finite Automata, and Polynomial-time Computations*.  
In Proceedings of the 14th Annual Symposium on theoretical Aspects of Computer Science (February 27 - March 01, 1997). R. Reischuk and M. Morvan, Eds. Lecture Notes In Computer Science, vol. 1200. Springer-Verlag, London, 117-128.  
<http://eccc.hpi-web.de/eccc-reports/1997/TR97-029/>