DTIME, NTIME, DSPACE, NSPACE, P, NP, PSPACE, NPSPACE, EXP, NEXP

Konstruovatelnost funkcie

Definícia 2.1.1. Funkcia $S(n)$ je páskovo konštruovateľná, ak existuje deterministický Turingov stroj ktorý na každom vstupe dĺžky n nepoužije na žiadnej pracovnej páske viac ako $S(n)$ políčok a na prvej páske použije práve $S(n)$ políčok.

Definícia 2.3.1. Funkcia $T(n)$ je časovo konštruovateľná, ak existuje deterministický T-stroj ktorý na každom vstupe dĺžky n vykoná práve $T(n)$ krokov.

kodovanie TS

$$1\,1\,1\,kód_1\,1\,1\,kód_2\,1\,1\,\cdots\,1\,1\,kód_h\,1\,1\,1$$

$$0^j1\,0^k1\,0^l1\,0^m1\,0^r1\,0^s1\,0^t$$

$$\delta(q_j, a_k, a_l) = (q_m, d_r, a_s, d_t)$$

kod_i = zapis jedneho prvku delta funkcie

Veta o pamatovej hierarchii

Veta [o pamäťovej hierarchii]. [1] *Nech* $\log(n) \leq S_1(n) = o(S_2(n))$, *kde* $S_2(n)$ *je páskovo konštruovateľná. Potom:*

$$DSPACE(S_1(n)) \subsetneq DSPACE(S_2(n))$$

spravime stroj M' ktory si vyhradi S2(n) miesta a toto miesto pocas behu neprekroci
skontroluje ci je vstup prefixovy kod nejakeho TS
simuluje vstup w na stroji s kodom w
simuluje 2^S2(n) krokov, teda ak w in DSPACE(S1(n)) tak sa bud rozhodol alebo sa uz nerozhodne
vratime negaciu
ak M' je DSPACE(S1) tak mame spor

Veta o časovej hierarchii

Veta [o časovej hierarchii]. *Nech* $n \leq T_1(n) = o(T_2(n)/\log T_1(n))$, *kde* $T_2(n)$ *je časovo konštruovateľná. Potom* $DTIME(T_1(n)) \subsetneq DTIME(T_2(n))$.

??? preco /log T1(n)

Veta [Savitch]. *Nech* $S(n) \geq \log(n)$ *je páskovo konštruovateľná, potom*

$$NSPACE(S(n)) \subseteq DSPACE(S^2(n)) .$$

budeme ratat funkciu f(C1, C2, t), ktora nam povie ci sa vieme dostat z konfiguracie C1 do C2 na najviac t krokov
kedze existuje iba d^S(n) konfiguracii tak akceptujuci vypocet musi trvat maximalne tolko krokov
f(C1, C2, t) = f(C1, C3, t/2) and f(C3, C2, (t+1)/2)
hlbka rekurzie je log2(d^S(n)) = S(n)*log2(d) = O(S(n)), pricom kazda uroven zabera S(n) pamate
pamatova zlozitost je O(S^2(n)) -> kompresia pasky na S^2(n)

Dalsie hierarchie

Veta 2.5.1. (a) $DTIME(T(n)) \subseteq NTIME(T(n)) \subseteq DSPACE(T(n))$ *pre každú časovo konštruovateľnú* $T(n)$.

(b) $DSPACE(S(n)) \subseteq NSPACE(S(n)) \subseteq \bigcup_c DTIME(c^{\log n + S(n)})$ *pre každú páskovo konštruovateľnú* $S(n)$.

NTIME(T(n)) ⊆ DSPACE(T(n))
budem lexikograficky generovat vsetky postupnosti instrukcii
NSPACE(S(n)) ⊆ DTIME exp(S(n))
existuje konecne vela konfiguracii, prechody medzi nimi su charakterizovane grafom, spravim DFS

Dôsledok 2.6.1.

$$DSPACE(\log n) \subseteq NSPACE(\log n) \subseteq P \subseteq$$

$$\subseteq NP \subseteq PSPACE = NPSPACE \subseteq EXP \subseteq NEXP$$

Poznámka 2.6.5. Nie je známe, či $DSPACE(\log n) \subsetneq NP$.

Poznámka 2.6.6. Podobne tiež nie je známe, či $P \subsetneq PSPACE$.

Poznámka 2.6.7. Naproti tomu sa vie, že $NSPACE(\log n) \subsetneq PSPACE$ a $P \subsetneq EXP$. Vyplýva to zo Savitchovej Vety a z Viet o pamäťovej a časovej hierarchii.

Translačná lema

Tvrdenie 2.7.1. *Ak* $NSPACE(n^4) \subseteq NSPACE(n^3)$, *potom:*

$$NSPACE(n^{20}) \subseteq NSPACE(n^{15}).$$

```
Zoberme lubovolny jazyk z NSPACE(n^20), existuje stroj M1 co ho akceptuje
Nafukneme vstup w an w#^i tak ze |w#^i| = |w|^5
Potom stroj M2 podobny M1 bude akceptovat v NSPACE(n^4)
Potom existuje stroj M3 co akceptuje w#^i v NSPACE(n^3)
Potom existuje stroj M4 co na vstupe w nafukne na w#^i a spusti M3, takze M4 je NSPACE(n^15)
```

Tvrdenie 2.7.2. $NSPACE(n^3) \subsetneq NSPACE(n^4)$.

Dôkaz. Nech by $NSPACE(n^4) \subseteq NSPACE(n^3)$. Aplikáciou translačnej lemy pre $f(n) = n^3$, $f(n) = n^4$, $f(n) = n^5$ postupne dostaneme:

$$\left.\begin{array}{r} NSPACE(n^{12}) \subseteq NSPACE(n^9) \\ NSPACE(n^{16}) \subseteq NSPACE(n^{12}) \\ NSPACE(n^{20}) \subseteq NSPACE(n^{15}) \end{array}\right\} \Rightarrow NSPACE(n^{20}) \subseteq NSPACE(n^9)$$

$$\left.\begin{array}{r} NSPACE(n^{20}) \subseteq NSPACE(n^9) \\ \text{Savitchova veta} \Rightarrow NSPACE(n^9) \subseteq DSPACE(n^{18}) \\ \text{Veta o pamäťovej hierarchii} \Rightarrow DSPACE(n^{18}) \subsetneq DSPACE(n^{20}) \end{array}\right\} \Rightarrow$$

$$\Rightarrow NSPACE(n^{20}) \subsetneq DSPACE(n^{20}) \subseteq NSPACE(n^{20}) \ \underline{spor!} \qquad \square$$

Translačná lema. *Nech $S_1(n)$, $S_2(n)$ a $f(n)$ sú páskovo konštruovateľné, $S_2(n) \geq n, f(n) \geq n$.*
Potom ak $NSPACE(S_1(n)) \subseteq NSPACE(S_2(n)) \Longrightarrow$
$\Longrightarrow NSPACE(S_1(f(n))) \subseteq NSPACE(S_2(f(n)))$.

Veta 2.7.3. *Ak $\varepsilon > 0$ a $r \geq 0$ potom*

$$NSPACE(n^r) \subsetneq NSPACE(n^{r+\varepsilon})$$

Veta [Gap Theorem]. *Pre každú rekurzívnu funkciu $g(n) \geq n$ existuje rekurzívna funkcia $S(n) \geq n$ taká, že platí:*

$$DSPACE(S(n)) = DSPACE(g(S(N))).$$

Poznámka 2.8.1. Podobné výsledky platia aj pre $NSPACE$, $DTIME$, $NTIME$.

Veta [Speedup Theorem]. *Pre každú rekurzívnu funkciu $f(n) \geq n^2$ existuje rekurzívny jazyk L taký, že pre ľubovoľný T-stroj M akceptujúci L v čase $T(n)$ existuje T-stroj M' tiež akceptujúci L, v čase $T'(n)$ tak, že $f(T'(n)) \leq T(n)$ pre skoro všetky n.*

Gap Theorem
existuje ocislovanie vsetkych strojov
definujeme funkciu W(n) (S(n) je matuce), tak, ze W(n) = m tak ze
pre vsetky stroje M1,...Mn na vsetkych vstupoch {0,1}^n plati ze S_Mk(w) <= m || S_Mk(w) > 2^W(n)

postup ako to spravit rekurzivne
paralelne simulujeme vsetky stroje na vsetkych vstupoch
povedzme ze robime krok z t-teho kroku simulacie na t+1
potom kazdy stroj co este nezastavil alebo sa nezacyklil spravime krok
znovu odsimulujeme t krokov daneho stroja pre dany vstup aby sme skontrolovali ci uz nenavstivil tuto konfiguraciu
pre vsetky n<=m<=t+1 **(vsimni si n<=m)** overime ci m splna danu podmienku pre vsetky stroje na vsetkych vstupoch, teda ci zastal/zacyklil a pouzil menej ako m alebo uz pouzil viac ako 2^m
toto musi nastat lebo bud vsetkych konecne vela strojov zastavi/zacykli alebo ak sa nezacyklia tak musia pouzit vela pamate (viac ako 2^m pre nejake m)

potom zoberme lubovolny jazyk L co patri DSPACE(2^W(n)), pre tento jazyk existuje stroj M co ho akceptuje a je rovny nejakemu Mk
platia dve veci
L in DSPACE(2^W(n)) => ∃n_0 ∀n>=n_0 S_Mk(n) ≤ 2^W(n)
definicia W(n) => ∀n>=k S_Mk(n) ≤ W(n) || S_Mk(n) > 2^W(n)
=> ∀n >= max(n_0, k) S_Mk(n) ≤ W(n) => L in DSPACE(W(n))

Definícia 3.0.1. $L \subseteq \Sigma^*$ je polynomiálne transformovateľný na $L_0 \subseteq \Sigma_0^*$, ak existuje polynóm $p(n)$ a deterministický T-stroj M s časovou zložitosťou $p(n)$, ktorý vstup $x \in \Sigma^*$ pretransformuje na vstup $y \in \Sigma_0^*$ taký, že $x \in L \Leftrightarrow y \in L_0$.

Definícia 3.0.2. L_0 je NP-úplný, ak $L_0 \in NP$ a každý $L \in NP$ je polynomiálne transformovateľný na L_0.

Veta [Cook-Levin]. *SAT je NP-úplný jazyk.*

Cook-Levin

Ideme SAT-om zapisat vypocet TS (jedna prepisovacia vstupna paska smerom doprava nekonecna)
Treba zarucit 4 podmienky - kazde policko kazdej konfiguracie obsahuje prave jeden znak (F), prva konfiguracia je pociatocna (G), posledna konfiguracia je v akceptacnom stave (I) a medzi kazdymi dvoma susednymi konfiguraciami je validny prechod (H)
validny prechod zabezpecime ak sa nezmeni symbol ktory je daleko od hlavy (H*) a hlava vykona jeden z posunov -1,0,1 (H**)

Konštrukcia F: $F := F_1 \wedge F_2 \wedge \cdots \wedge F_h, \quad h = (p(n)+1)(p(n)+2)$

$$F_j := \Big(\bigvee_{s \in \Sigma' \cup Q}^{>=1} y_{j,s}\Big) \wedge \neg\Big(\bigvee_{\substack{s \neq r \\ s,r \in \Sigma' \cup Q}}^{<2} (y_{j,s} \wedge y_{j,r})\Big), \quad j = 1, 2, \ldots, h$$

$$G := y_{1,\not\in} \wedge y_{2,q_0} \wedge y_{3,x_1} \wedge y_{4,x_2} \wedge \cdots \wedge y_{n+2,x_n} \wedge y_{n+3,B} \wedge \cdots \wedge y_{p(n)+2,B}$$

$$I := y_{t,q_F} \vee y_{t+1,q_F} \vee \cdots \vee y_{t+p(n)+1,q_F}, \quad t = p(n)(p(n)+2)+1$$

Pre každý reťazec rqs ($r, s \in \Sigma'$, $q \in Q$) nech:

$K_{rqs} := \{tuv \mid t, u, v \in \Sigma' \cup Q,\ tuv$ môže vzniknúť z rqs v 1 kroku v súlade s δ (pozri (**))$\}$

Nech $H_i^* = \bigwedge_{i(p(n)+2)+1 \le j \le (i+1)(p(n)+2)} \left(\bigwedge_{r,s,t \in \Sigma'} (y_{j-1,r} \wedge y_{j,s} \wedge y_{j+1,t} \Rightarrow y_{j+p(n)+2,s}) \right)$

$$H_i^{**} = \bigwedge_{\cdots \le j \le \cdots} \left(\bigwedge_{\substack{r,s \in \Sigma' \\ q \in Q}} \left(y_{j-1,r} \wedge y_{j,q} \wedge y_{j+1,s} \Longrightarrow \bigvee_{tuv \in K_{rqs}} (y_{j+p(n)+1,t} \wedge y_{j+p(n)+2,u} \wedge y_{j+p(n)+3,v}) \right) \right)$$

$$i = 0, 1, \cdots, p(n)-1$$

Potom $H = H_0^* \wedge H_1^* \wedge \cdots \wedge H_{p(n)-1}^* \wedge H_0^{**} \wedge H_1^{**} \wedge \cdots \wedge H_{p(n)-1}^{**}$ je výraz, ktorý kontroluje, či

Definícia 4.1.1. **NP-optimalizačný problém** A je daný cieľom (t.j. *min* alebo *max*), polynomiálne ohraničenou reláciou $R \subseteq \Sigma^* \times \Sigma^*$ (t.j. pre R existuje polynóm p taký, že ak $(x, y) \in R$, potom $|y| \leq p(|x|)$) a hodnotovou funkciou $m : \Sigma^* \times \Sigma^* \to \mathbb{N}$, pričom R aj m sú vypočítateľné deterministicky v polynomiálnom čase.

Presnejšie $\{x\#y \mid (x, y) \in R\} \in P$ a pre dvojicu $(x, y) \in R$ možno hodnotu funkcie $m(x, y)$ vypočítať deterministicky v čase $q(|x| + |y|)$, kde q je nejaký polynóm.

Definícia 4.1.2. **Konštrukčný problém**: Pre dané x nájdi/zostroj (ak existuje) y také, že $(x, y) \in R$, pričom:

$$m(x, y) = \min_{y' \in \Sigma^*} \{m(x, y') \mid (x, y') \in R\} \text{ pre cieľ } min \text{ a podobne:}$$

$$m(x, y) = \max_{y' \in \Sigma^*} \{m(x, y') \mid (x, y') \in R\} \text{ pre cieľ } max.$$

Definícia 4.1.3. **Hodnotový problém**: Pre dané x urči (ak existuje):

$$\min_{y' \in \Sigma^*} \{m(x, y') \mid (x, y') \in R\} \text{ pre cieľ } min,$$

$$\max_{y' \in \Sigma^*} \{m(x, y') \mid (x, y') \in R\} \text{ pre cieľ } max.$$

Definícia 4.1.4. **Rozhodovací roblém**: Pre dané x a dané $k \in \mathbb{N}$ zisti, či:

$$\exists y : (x, y) \in R \wedge m(x, y) \leq k \text{ pre cieľ } min,$$

$$\exists y : (x, y) \in R \wedge m(x, y) \geq k \text{ pre cieľ } max.$$

Veta 4.1.2. *Nech A je ľubovoľný NP-optimalizačný problém s cieľom min[1], reláciou R a hodnotovou funkciou m, ktorého rozhodovací problém:*

$$L = \{x\#d(k) \mid x \in \Sigma^*, k \in \mathbb{N}, \exists y : (x,y) \in R \wedge m(x,y) \leq k\}$$

je NP-úplný. Nech možno L akceptovať deterministicky v čase $T(n)$. Potom konštrukčný problém pre A možno riešiť deterministicky v čase $r(n)T(s(n))$ pre vhodné polynómy r, s. [2] [3]

Kedze hodnotova funkcia je polynomialne ohranicena, vieme binarne vyhladat riesenie hodnotoveho problemu. Toto vieme v case T(s(n))*r(n).
Teraz zostrojime optimalne riesenie. Jazyk

$L' = \{x\#d(k)\#z | x \in \Sigma^*, k \in \mathbb{N}, \exists y : (x,y) \in R \wedge m(x,y) \leq k$ a z je prefix reťazca $y\}$.

je NP. Potom ho vieme polynomialne transformovat na L, ktory je NP-uplny. Potom sa staci pytat na pritomnost stale dlhsich a dlhsich slov s prefixom x#d(k)# v L' az kym sa slovo nebude dat predlzit a suffix 'z' tohoto slova bude riesenim daneho konstrukcneho problemu.

Dôsledok 4.1.1. (a) Veta 4.1.2 platí aj keď predpoklady „L je NP-úplný" a „L možno akceptovať deterministicky v čase $T(n)$" nahradíme predpokladom „nejaký NP-úplný jazyk L_0 možno akceptovať deterministicky v čase $T(n)$".

(b) Ak $P = NP$, potom konštrukčný problém každého NP-optimalizačného problému možno riešiť deterministicky v polynomiálnom čase.

Veta 4.1.3. *Nech A je ľubovoľný NP-optimalizačný problém, ktorého rozhodovací problém L možno akceptovať v deterministickom čase $T(n)$ (L nemusí byť NP-úplný). Potom hodnotový problém pre A možno riešiť deterministicky v čase $r(n)T(s(n))$ pre vhodné polynómy r, s.*

Definícia 4.2.2. Problém A je *dobre* aproximovateľný, ak je α-aproximovateľný pre každé $\alpha > 1$ $(\alpha \to 1)$.

Definícia 4.2.3. Problém A je *neaproximovateľný*, ak nie je α-aproximovateľný pre žiadne $\alpha > 1$ $(\alpha \to \infty)$.

Veta 4.2.1. *Ak $P \neq NP$, potom existujú NP-optimalizačné problémy A, B a C také, že:*

(i) A je dobre aproximovateľný, ale jeho rozhodovací problém nepatrí do P.

(ii) B je α-aproximovateľný (pre nejaké $\alpha > 1$), ale nie je dobre aproximovateľný.

(iii) C je neaproximovateľný.

Dôkaz. A, B a C sú rovnaké, až na hodnotovú funkciu a sú definované takto: cieľ je *max*,

$$R = \{(x, y) \mid x \text{ je graf, } y \text{ je ľub. postupnosť všetkých vrcholov grafu } x \text{ (bez opakovania)}\}$$

a hodnotová funkcia m_A resp. m_B resp. m_C pre A resp. B resp. C je definovaná takto:

$$m_A(x, y) = \begin{cases} |x| & \text{, ak } y \text{ je hamiltonovská kružnica grafu } x \\ |x| - 1 & \text{, inak} \end{cases}$$

$$m_B(x, y) = \begin{cases} 2 & \text{, ak } y \text{ je hamiltonovská kružnica grafu } x \\ 1 & \text{, inak} \end{cases}$$

$$m_C(x, y) = \begin{cases} |x| & \text{, ak } y \text{ je hamiltonovská kružnica grafu } x \\ 1 & \text{, inak} \end{cases}$$

Veta 4.2.2. *Ak $P \neq NP$, potom NP-optimalizačný* „problém obchodného cestujúceho" (POC) *je neaproximovateľný.*

Ak by POC bol aproximovatelny pre nejake α, potom pre graf s cenami hran nastavenymi nasledovne

$$cena(i,j) = \begin{cases} 1 & , \text{ak } (i,j) \in E \\ \alpha|V| + 1 & , \text{ak } (i,j) \notin E \end{cases}$$

bude riesenie optimalneho POC bud rovne |V| (ak existuje HAM) alebo >= (1+α)|V| >= α|V|+1 (ak neexistuje). V pripade ze HAM existuje, aproximacny algoritmus musi najst ohodnotenie <=α|V|. Ak HAM neexistuje, optimalne ohodnotenie je >=α|V|+1 > α|V|. Preto aproximacnym algoritmom budeme vediet rozlisit tieto dva pripady a riesit HAM, co je spor.

Veta 4.2.3. *Ak $P \neq NP$, potom:*

(1) „Maximálna klika grafu" *(jej nájdenie) a* „Najdlhšia cesta v grafe medzi dvoma danými vrcholmi" *sú neaproximovateľné NP-optimalizačné problémy.*

(2) „Bin-packing" *(problém kontajnerov) je* $\frac{3}{2}$*-aproximovateľný, ale nie je* $(\frac{3}{2}-\varepsilon)$*-aproximovateľný pre žiadne* $\varepsilon > 0$ *(pokiaľ* $P \neq NP$*).*

(3) „POC s trojuholníkovou nerovnosťou" *je* $\frac{3}{2}$*-aproximovateľný ale nie je dobre aproximovateľný.*

(4) „0-1 Knapsack" *(problém batoha) je dobre aproximovateľný NP-optimalizačný problém.*

Miller-Rabinov test:

Vstup: nepárne $b > 2$, $r \in \mathbb{N}$.

Nech $[2^t \mid (b-1)] \wedge [2^{t+1} \nmid (b-1)]$

for $i \leftarrow 1$ **to** r **do**

begin

vyber náhodne celé a $(1 \leq a \leq b-1)$

1. **if** $x^2 \equiv 1 \pmod{b}$ **and** $x \not\equiv \pm 1 \pmod{b}$

pre nejaké $x \in \{a^{(b-1)/2}, a^{(b-1)/4}, a^{(b-1)/8}, \ldots, a^{(b-1)/2^t}\}$

then return „b **určite nie je prvočíslo**" (s absolútnou istotou)

2. **if** $a^{b-1} \not\equiv 1 \pmod{b}$

then return „b **určite nie je prvočíslo**" (s absolútnou istotou) [1]

end

3. **return** „b **je prvočíslo**" (s pravdepodobnosťou omylu nanajvýš 2^{-r}).

Veta [Miller-Rabin]. *Ak b nie je prvočíslo, potom náhodne vybrané číslo a $(1 \leq a \leq b-1)$ spĺňa podmienku v príkaze* **1.** *alebo* **2.** *s pravdepodobnosťou aspoň $\frac{1}{2}$.*

```
Najdeme najvacsie T take ze 2^T|(B-1) (B je neparne)
pre nahodne cislo A ho skusame umocnit na stupajuce mocniny (B-1)/2^i az po (B-1)/2
Zacneme s A^[(B-1)/2^T] - ak je to 1 alebo -1 tak pravdepodobne ide o prvocislo
Pokracujeme umocnovanim ^2
Ak je rovne -1, pravdepodobne ide o prvocislo, ak je rovne 1 urcite nejde o prvocislo
Ak prideme na koniec a este sme nezistili ze ide o prvocislo tak urcite nejde o prvocislo (A^(B-1)=1,
preto A^(B-1)/2=-1 a zistili by sme to v poslednej iteracii)
```

If p is an odd prime, and $p - 1 = 2^s\, d$ with d odd, then for every a prime to p, either $a^d \equiv 1 \bmod p$, or there exists t such that $0 \leq t < s$ and $a^{d2^{\wedge}t} \equiv -1 \bmod p$.

Pravdepodobnostné generovanie veľkých prvočísel

1. vygeneruj nahodne cislo
2. over ci je prvocislo
3. ???
4. profit / repeat

$$\lim_{n\to\infty} \frac{P(n)}{n/\ln n} = 1$$

RSA

(1) Vyber náhodné veľké (dekadicky 100-ciferné) prvočísla p, q.

(2) Vypočítaj $n = pq$, $\phi(n) = (p-1)(q-1)$.

(3) Vyber nepárne číslo e nesúdeliteľné s $\phi(n)$. Presnejšie, pomocou procedúry `Euklid` (pozri kapitolu *Modulárna aritmetika* (6)) nájdi $e \in \{3, 5, 7, \dots\}$, a celé x, y také, že $1 = \phi(n)x + ey$.

(4) Vypočítaj $d = e^{-1} \bmod \phi(n)$, t.j. prvok inverzný k e v zvyškovej triede čísel modulo $\phi(n)$, presnejšie: $[d \leftarrow y;$ **if** $y < 0$ **then** $d \leftarrow y + \phi(n)]$. Potom pre d platí $1 = (e \cdot d) \bmod \phi(n)$, $d \geq 0$.

(5) Nech $D = Z_n = \{0, 1, \dots, n-1\}$. Potom:

- $P_U(M) = M^e \bmod n$,
- $S_U(M) = M^d \bmod n$,

Presny algoritmus pre 0-1 knapsack problem

dynamicke programovanie

postupne pre vsetky itemy upravujeme tabulku

kluc stavu je value a hodnota je minimalna dosiahnutelna hmotnost (/ kluc stavu je hmotnost a hodnota je max dosiahnuta value)

hodnotovy problem riesime max{v | W(n, v) <= W}

konstrukcny problem riesime tak ze sa vratime o krok spat a zistime overime ci sme sa pridanim itemu dostali do aktualneho stavu

Θ(n^2 V) = Θ(n^2 * 2^Θ(n)) = Θ(|x| * 2^Θ(√|x|)) kedze na vstupe mozu byt velmi velke cisla

Aproximacny algoritmus pre 0-1 knapsack problem

1 < α <= 2, V = max(v_i), d = trunc((α - 1) V / (n + 1)), ak d=0 tak d=1

predel vsetky hodnoty itemov d a zaokruhli nadol

spusti presny algoritmus

casova zlozitost n^2*V -> n^2*V/d = n^3 / (α - 1) = polynomialna

dokaz aproximacie netreba vediet

Veta. *NP-optimalizačný 0-1 knapsack problém je dobre aproximovateľný.*

Pravdepodobnostne algoritmy

procedura brand vrati 0/1 -> vieme generovat lubovolne m-bitove cisla, cize napriklad cisla z <0,1) s lubovolnou presnostou

Pr[A akceptuje x] = suma pravdepodobnosti akceptacnych vypoctov, Pr[A akceptuje x] = 1 - Pr[A zamietne x]

Definícia 2. *Hovoríme, že pravdepodobnostný algoritmus A akceptuje jazyk $L \subseteq \Sigma^*$ s chybou ϵ, $(0 \leq \epsilon < 1/2)$, ak $\forall x \in \Sigma^*$ platí: Ak $x \notin L$, potom $Pr[A \text{ akceptuje } x] \leq \epsilon$ a ak $x \in L$, potom $Pr[A \text{ zamietne } x] \leq \epsilon$.*

Veta (o vylepšovaní). *Nech A je pravdepodobnostný algoritmus akceptujúci jazyk L v čase $T(n)$ s chybou ϵ, $(0 < \epsilon < 1/2)$. Potom pre každé ϵ', $(0 < \epsilon' < \epsilon)$, existuje pravdepodobnostný algoritmus A' akceptujúci L v čase $O(T(n))$ s chybou ϵ'.*

dokaz ???

$$\frac{1}{2}(4(1-\epsilon)\epsilon)^{m/2} \leq \epsilon'. \qquad S_x = \Sigma_{j=0}^{\lfloor m/2 \rfloor} \binom{m}{j} (1-\epsilon_x)^j \epsilon_x^{m-j}$$

$$\begin{aligned} S_x &\leq \sum_{j=0}^{\lfloor m/2 \rfloor} \binom{m}{j} (1-\epsilon_x)^j \epsilon_x^{m-j} ((1-\epsilon_x)/\epsilon_x)^{m/2-j} \qquad (\text{podľa } (4)) \\ &= (1-\epsilon_x)^{m/2} \epsilon_x^{m/2} \Big(\sum_{j=0}^{\lfloor m/2 \rfloor} \binom{m}{j} \Big) \\ &= \frac{1}{2}(2^2(1-\epsilon_x)\epsilon_x)^{m/2} \qquad (\text{lebo } \sum_{j=0}^{\lfloor m/2 \rfloor} \binom{m}{j} = 2^m/2) \\ &\leq \frac{1}{2}(4(1-\epsilon)\epsilon)^{m/2} \qquad (\text{podľa } (5)). \end{aligned}$$

Veta. *Nech $L \subseteq \{0,1\}^*$ je ľubovoľný jazyk, pre ktorý existuje pravdepodobnostný algoritmus, ktorý akceptuje L v polynomiálnom čase s chybou ϵ, $(0 < \epsilon < 1/2)$. Potom existuje polynóm $p(n)$ a pravdepodobnostný algoritmus A s polynomiálnou časovou zložitosťou tak, že pre každé n existuje postupnosť $b^n = b_1^n, b_2^n, \ldots, b_{p(n)}^n$, kde $b_i^n \in \{0,1\} \forall i$, taká, že $\forall x \in \{0,1\}^n$ platí: Výpočet algoritmu A na x, v ktorom výsledok i-teho volania procedúry $brand$ je b_i pre každé $i = 1, 2, \ldots, p(n)$, je akceptujúci, ak $x \in L$ a je zamietajúci, ak $x \notin L$, (t.j., A poznajúc b^n neurobí chybu na žiadnom $x \in \{0,1\}^n$).*

Pre L existuje algoritmus A s polynomialnou casovou zlozitostou p(n) a chybou = 1/7
Na vstupe velkosti n vykoname 2n+1 vypoctov a akceptujeme/zamietame podla vacsiny
Upravime vypocet tak aby vzdy vykonal p(n) operacii brand (aj keby mohol skoncit skor)
Z vety o vylepsovani vieme ze chybnych vypoctov je najviac 2^p(n) * (½)^(n+3/2)
Vsetkych chybnych vypoctov na vsetkych moznych vstupov je dokopy 2^n * 2^p(n) * (½)^(n+3/2) = 2^p(n) * (½)^(3/2) < 2^p(n)
Vsetkych postupnosti vysledkov funkcie brand je 2^p(n) a teda musi existovat taka postupnost ktora nie je sucastou ziadneho chybneho vypoctu
Na prednaske to bolo vysvetlovane aj cez tabulku kde v riadkoch su vstupy a v stlpcoch su postupnosti vysledkov brand

Definícia 5. *BPP je trieda jazykov, ktoré možno akceptovať pravdepodobnostnými Turingovými strojmi v polynomiálnom čase s nejakou chybou* $0 \leq \epsilon < \frac{1}{2}$.

Veta. $P \subseteq BPP \subseteq PSPACE.$

P ⊆ BPP zjavne

BPP ⊆ PSPACE
Nech L bolo z BPP, potom L je akceptovane strojom A v polynomialnom case
Vytvorime stroj ktory bude akceptovat L a zjavne bude z PSPACE
Na paske si bude v lexikografickom poradi generovat vsetky postupnosti vysledkov volani brand a nasledne simulovat A na zaklade danych rozhodnuti
Priebezne si pocita pocet akceptacii a nakoniec sa rozhodne podla vacsiny
Treba si davat pozor aby dlzka postupnosti brandov presne zodpovedala dlzke vypoctu

Poznámka:
- vieme, že platí $P \subseteq NP \subseteq PSPACE$
- nie je známy žiadny vzťah medzi NP a BPP
- nie je známe, či $P = BPP$ alebo, či $BPP = PSPACE$

Utajene riesenie NP-uplneho problemu

Chceme dokazat ze vieme vrcholovo ofarbit graf G tromi farbami

Zpermutujeme farby, pre kazdy vrchol vygenerujeme RSA kluce, pre kazdy vrchol zvolime nahodne cislo x_i od 0 po n/3, zasifrujeme 3*x_i+farba_i

Verejne kluce a zasifrovane hodnoty zverejnime

Protihrac sa moze spytat na lubovolnu hranu

Zverejnime sukromne kluce pre dane dva vrcholy

Protihrac overi ze farby vrcholov su rozdielne

Ak sme graf neofarbili spravne, je sance aspon 1/|E| ze na to protihrac pride, teda po k*|E| kolach je sanca ze na to neprisiel

$$\left(\frac{|E|-1}{|E|}\right)^{k|E|} = \left(1 - \frac{1}{|E|}\right)^{k|E|} \leq e^{-k}$$

Kedze vsetko je strasne nahodne a zamiesane a dozvie sa informaciu iba o jednej hrane pred tym ako sa vsetko zamiesa tak je to bezpecne

$PSPACE = \bigcup_{k>0} DSPACE(n^k)$.

Definícia. *Jazyk L_0 je PSPACE-úplný, ak $L_0 \in PSPACE$ a každý $L \in PSPACE$ je polynomiálne tranformovateľný na L_0 .*

Definícia. *Booleovská formula s kvantifikátormi je úplne kvantifikovaná, ak každá jej premenná je v oblasti pôsobnosti niektorého kvantifikátora.*

Poznámka. Každú úplne kvantifikovanú formulu možno napísať v tvare $Q_1 x_1 Q_2 x_2 \ldots Q_n x_n \ \phi(x_1, x_2, \ldots, x_n)$, kde každé Q_i je nejaký kvantifikátor a $\phi(x_1, \ldots, x_n)$ neobsahuje žiadny kvantifikátor.

Veta. *Jazyk TQBF je PSPACE-úplný.*

je PSPACE
rekurzivne sa volaj a skusaj vsetky ohodnotenia premennych
pamatova zlozitost je linearna

je PSPACE-uplny
podobne ako Cook-Levin a Savitchova veta
ideme transformovat TS na vstupe dlzky n na TQBF
kedze nechceme aby sa zacyklil tak dlzka vypoctu musi byt najviac 2^(r*S(n)) pre nejake r

$$\Phi^1(\tilde{x},\tilde{y}) = ((H_i^* \wedge H_i^{**}) \vee (\tilde{x} \equiv \tilde{y})) \wedge \left(\bigwedge_{i(S(n)+2)+1 \leq j \leq (i+2)(S(n)+2)} F_j \right)$$

$$\Phi^{2l}(\tilde{u},\tilde{v}) = \exists \tilde{z}(\Phi^l(\tilde{u},\tilde{z}) \wedge \Phi^l(\tilde{z},\tilde{v}))$$ toto je prilis dlhe, pocet premennych narasta exponencialne

$$\Phi^{2l}(\tilde{u},\tilde{v}) = \exists \tilde{z} \forall \tilde{x} \forall \tilde{y}((((\tilde{x} \equiv \tilde{u}) \wedge (\tilde{y} \equiv \tilde{z})) \vee ((\tilde{x} \equiv \tilde{z}) \wedge (\tilde{y} \equiv \tilde{v}))) \Rightarrow \Phi^l(\tilde{x},\tilde{y}))$$

dokaz spravnosti indukciou na el

Hra BF

- daná je booleovská formula $\phi(x_1, \ldots, x_n)$, (bez kvantifikátorov $\forall$, $\exists$)
- hru hrajú striedavo dvaja hráči H_1 a H_2; začína H_1
- hráči postupne a striedavo určujú hodnoty $b_1, b_2, \ldots, b_n$ premenným $x_1, x_2, \ldots, x_n$ (v každom kroku hry jednu b_i; H_1 nepárnym premenným a H_2 párnym)
- hru vyhral hráč H_1 *iff* $\phi(b_1, \ldots, b_n) = 1$

Veta. *Jazyk BF je PSPACE-úplný.*

TQBF pretransformujeme na BF tak ze vyjmeme kvantifikatory na zaciatok a pridame always true premenne tak aby sa kvantifikatory striedali

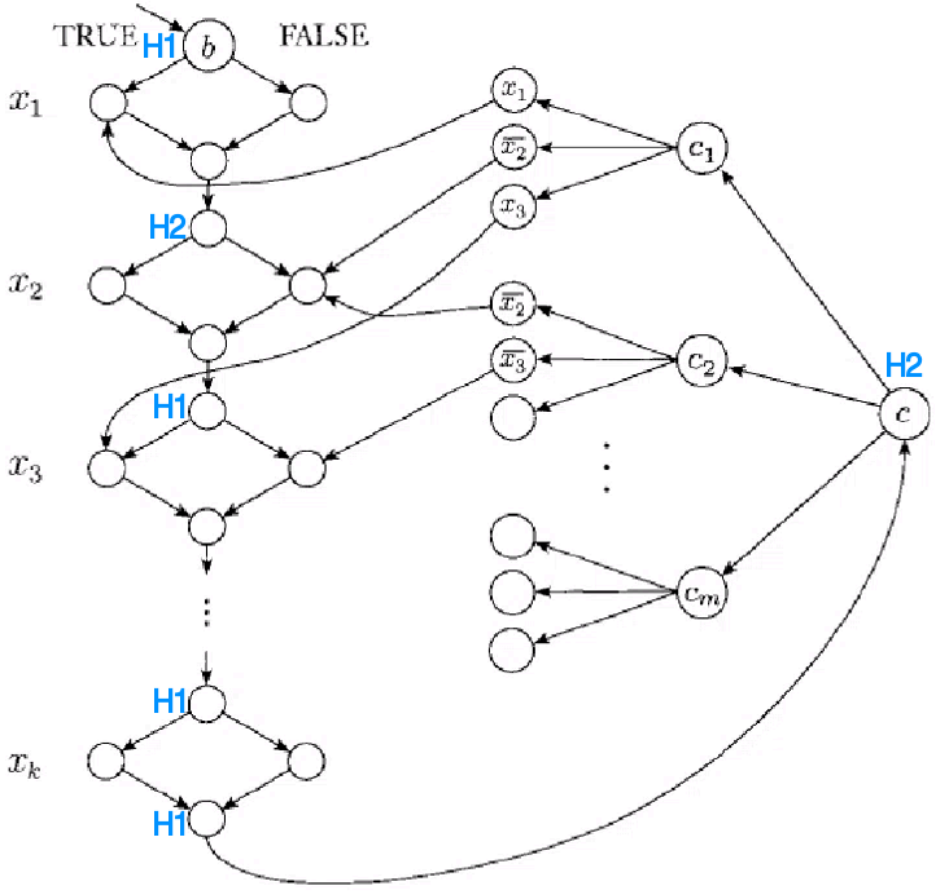
$$\exists y_1 \forall x_1 \exists x_2 \forall y_2 \exists x_3 \forall x_4 \;\; \phi(x_1, \ldots, x_4) \wedge \underbrace{(y_1 \vee \bar{y}_1)}_{1} \wedge \underbrace{(y_2 \vee \bar{y}_2)}_{1}$$

???

Hra GG

- daný je orientovaný graf G a jeho vrchol b
- hru hrajú striedavo dvaja hráči H_1 a H_2; začína H_1 vo vrchole b
- hráči postupne a striedavo navštevujú (z momentálne navštíveného vrchulu) ešte nenavštívené susedné vrcholy, (v každom kroku jeden vrchol)
- hru prehral hráč, ktorý už nemôže navštíviť žiadny ešte nenavštívený vrchol

Veta. *Jazyk GG je PSPACE-úplný.*



BF polynomialne transformujeme na GG
formula bude SAT
H1 a H2 v lavej casti hodnotia premenne
H2 urci ktoru disjunkciu chce aby H1 dokazal (ak je formula neplatna, vie najst taku disjunkciu ktora neplati)
H1 potom vyberie literal ktory disjunkciu riesi
H2 je na tahu a ak bol H1 uspesny tak je prislusne miesto v lavej casti uz obsadene a teda H2 hru prehral

Rekurzivne funkcie

V tejto podkapitole budeme uvažovať len funkcie typu $f : N^n \to N$, $(N = \{0, 1, 2, \ldots\})$, pričom funkcia f nemusí byť totálna.

Označenia.

(a) *Funkcia* I_m^n, $(1 \leq m \leq n)$, *je* n*-árna funkcia, pre ktorú platí:* $I_m^n(x_1, \ldots, x_n) = x_m$ *pre všetky* $x_1, \ldots, x_n \in N$.

(b) *Funkcia* s *je unárna funkcia, pre ktorú platí:* $s(x) = x + 1$, $\forall x \in N$. *(Funkcia* s *je nasledovník.)*

(c) *Symbol* 0 *označuje (okrem iného aj) nula-árnu funkciu* s *konštantnou hodnotou* 0.

Definícia. *Nech* f *je* n*-árna funkcia a* $g, f_1, \ldots, f_n$ *sú* m*-árne funkcie. Funkcia* g *vznikne* ***skladaním*** *z funkcií* $f, f_1, \ldots, f_n$, *ak pre všetky* $x_1, \ldots, x_m \in N$ *platí:*

$$g(x_1, \ldots, x_m) = f\left(f_1(x_1, \ldots, x_m), \ldots, f_n(x_1, \ldots, x_m)\right).$$

g(x1, x2, x3) = f(x1, f1(x1), f2(x1, x2), f3(x1, x2, x3)) nezodpoveda specifikacii skladania funkcii, ale je to prehladnejsie a je to lahke opravit

Definícia. *Nech h je n-árna funkcia, g je $(n+2)$-árna funkcia a f je $(n+1)$-árna funkcia. Funkcia f vznikna z funkcií g a h operáciou* ***primitívnej rekurzie****, ak pre všetky $x_1, \ldots, x_n, y \in N$ platí:*

$$f(0, x_1, \ldots, x_n) = h(x_1, \ldots, x_n),$$
$$f(y+1, x_1, \ldots, x_n) = g(y, f(y, x_1, \ldots, x_n), x_1, \ldots, x_n).$$

Definícia. *Funkcia f je* ***primitívne rekurzívna****, ak vznikne z funkcí 0, s a I_m^n konečným počtom operácií skladania funkcií a primitívnej rekurzie. (Primitívne rekurzívne funkcie sú preto totálne.)*

Veta 1. *Konštantná funkcia $K_0(x) = 0$, $\forall x \in N$, je primitívne rekurzívna.*

$K_0(0) = 0 = h$,
$K_0(y+1) = 0 = K_0(y) = I_2^2(y, K_0(y)) = g(y, K_0(y))$.

Dôsledok 1. *Nech $j \geq 0$. Potom konštantná funkcia $K_j(x) = j$, $\forall x \in N$, je primitívne rekurzívna.*

Veta 2. *Funkcie F_1 až F_7 sú primitívne rekurzívne.*

$$F_1(x,y) = x + y$$

$$F_2(x,y) = x \cdot y$$

$$F_3(x,y) = y^x$$

$$F_4(x) = sg(x) = \begin{cases} 0, & ak\ x = 0 \\ 1, & inak \end{cases}$$

$$F_5(x) = \overline{sg}(x) = 1 - sg(x)$$

$$F_6(x,y) = y \dot{-} x = \begin{cases} 0, & ak\ y < x \\ y - x, & ak\ y \geq x \end{cases}$$

$$F_7(x,y) = |x - y| = (x \dot{-} y) + (y \dot{-} x)$$

Príklad 1: Funkcia $f(y,x) = y + x$ vznikne operáciou primitívnej rekurzie z funkcií $g(y,z,x) = z + 1$ a $h(x) = x$, lebo platí:

$$f(0,x) = x = h(x),$$
$$f(y+1,x) = y + 1 + x = f(y,x) + 1 = g(y, f(y,x), x).$$

F2, F3 obdobne

F4:

$$g(x,y) = K_1(I_1^2(x,y)) = 1,$$

$sg(0) = 0$, (nula vpravo je nula-árna funkcia 0),
$sg(x+1) = 1 = g(x, sg(x))$.

Pre F_6: (Idea.) Podobne, ako pre F_4 alebo F_5 možno dokázať, ze funkcie $F'_6(x) = x \dot{-} 1$ a F_6 sú primitívne rekurzívne, lebo platí:
$F'_6(0) = 0$, (nula vpravo je nula-árna funkcia z bodu (c) vyššie),
$F'_6(x+1) = x = I^2_1(x, F'_6(x))$ a
$F_6(0, y) = y = I^1_1(y)$,
$F_6(x+1, y) = (y \dot{-} x) \dot{-} 1 = F'_6(I^3_2(x, F_6(x, y), y))$.

Veta 3. *Nech $h(y, x_1, \ldots, x_n)$ je primitívne rekurzívna funkcia. Potom aj funkcie*

$$f_1(y, z, x_1, \ldots, x_n) = \sum_{i=z}^{y+z} h(i, x_1, \ldots, x_n),$$

$$f_2(y, z, x_1, \ldots, x_n) = \prod_{i=z}^{y+z} h(i, x_1, \ldots, x_n),$$

sú primitívne rekurzívne.

$$\begin{aligned} f_1(0, z, x_1, \ldots, x_n) &= h(z, x_1, \ldots, x_n) \\ f_1(y+1, z, x_1, \ldots, x_n) &= \sum_{i=z}^{y+z} h(i, x_1, \ldots, x_n) + h(y+z+1, x_1, \ldots, x_n) \\ &= f_1(y, z, x_1, \ldots, x_n) + h(y+z+1, x_1, \ldots, x_n) \\ &= g(y, f_1(y, z, x_1, \ldots, x_n), z, x_1, \ldots, x_n), \end{aligned}$$

$$g(y, u, z, x_1, \ldots, x_n) = u + h(y+z+1, x_1, \ldots, x_n) = F_1(u, h(s(F_1(y, z)), x_1, \ldots, x_n))$$

Definícia. *Čiastočná funkcia $f(x_1, \ldots, x_n)$ vznikne z čiastočnej funkcie $g(y, x_1, \ldots, x_n)$* ***minimalizáciou****, zapisujeme*

$$f(x_1, \ldots, x_n) = \mu_y(g(y, x_1, \ldots, x_n) = 0),$$

ak prc všctky $x_1, \ldots, x_n$ platí: $f(x_1, \ldots, x_n) = y$ iff kcd̀ pre všetky $z < y$ jc $g(z, x_1, \ldots, x_n)$ definovaná a kladná a $g(y, x_1, \ldots, x_n) = 0$. ($y$ je prvý nulový bod funkcie g vzhl̆adom na $x_1, \ldots, x_n$.)

Ak naviac f a g sú totálne, hovoríme, že f vznikne z g ***regulárnou minimalizáciou****.*

Poznámka. Operácie minimalizácie nezachováva totálnosť a ani primitívnu rekurzívnosť.

Príklad : Nech $f(0) = 0$ a nech $f(x)$ nie je definivaná pre žiadne $x > 0$. Pre f platí: $f(x) = \mu_y(I_2^2(y, x) = 0)$.

Definícia. *Funkcia f je* ***rekurzívna****, ak f vznikne z funkcií 0, s a I_m^n konečným počtom operácií skladania funkcií, pritívnej rekurzie a regulárnej minimalizácie. (Rekurzívne funkcie sú preto totálne.)*

Poznámka. Každá primitívne rekurzívna funkcie je rekurzívna.

Definícia. *Čiastočná funkcia f je* **čiastočne rekurzívna**, *ak f vznikne z funkcií 0, s a I_m^n konečným počtom operácií skladania funkcií, primitívnej rekurzie a minimalizácie.*

Veta 4. *Nech funkcie $g(y, x_1, \ldots, x_n)$ a $h(x_1, \ldots x_n)$ sú primitívne rekurzívne a nech pre každé $x_1, \ldots, x_n$ existuje $u \leq h(x_1, \ldots, x_n)$ také, že $g(u, x_1, \ldots, x_n) = 0$. Potom je funkcia $f(x_1, \ldots, x_n) = \mu_y(g(y, x_1, \ldots, x_n) = 0)$ primitívne rekurzívna.*

$$f(x_1, \ldots, x_n) = \sum_{y=0}^{h(x_1,\ldots,x_n)} sg(\prod_{i=0}^{y} g(i, x_1 \ldots, x_n))$$

mne sa paci ked je mul(sg(...)) namiesto sg(mul(...))

Veta 5. *Funkcie F_8 až F_{15} sú primitívne rekurzívne.*

$$F_8(x,y) = \lfloor x/y \rfloor, \qquad (\lfloor x/y \rfloor = 0,\ \text{ak } y = 0)$$

$$F_9(x,y) = x \bmod y, \qquad (x \bmod y = x,\ \text{ak } y = 0)$$

$$F_{10}(x,y) = \mathrm{div}(x,y) = \begin{cases} 1, & \text{ak } y \text{ delí } x \\ 0, & \text{inak} \end{cases}$$

$$F_{11}(x) = \mathrm{nd}(x) = \begin{cases} \text{počet deliteľov čísla } x, & \text{ak } x \neq 0 \\ 0, & \text{inak} \end{cases}$$

$$F_{12}(x) = \mathrm{chp}(x) = \begin{cases} 0, & \text{ak } x \text{ je prvočíslo} \\ 1, & \text{inak} \end{cases}$$

$$F_{13}(x) = \pi(x) = \text{ počet prvočísel nepresahujúcich } x$$

$$F_{14}(x) = \mathrm{p}(x) = x\text{-té prvočíslo } (t.j., \mathrm{p}(0) = 2, \mathrm{p}(1) = 3, \mathrm{p}(2) = 5, \ldots)$$

$$F_{15}(x,y) = \mathrm{ex}(x,y) = \text{exponent prvočísla } \mathrm{p}(x) \text{ v rozklade čísla } y \text{ na prvočísla, } (\mathrm{ex}(x,y) = 0 \text{ pre } y = 0)$$

$$\lfloor x/y \rfloor = sg(y) \sum_{i=1}^{(x \dot{-} 1)+1} \overline{sg}(i \cdot y \dot{-} x)$$

$$x \bmod y = x \dot{-} y \cdot \lfloor x/y \rfloor, \; \mathrm{div}(x, y) = \overline{sg}(x \bmod y),$$

$$\mathrm{nd}(x) = sg(x) \sum_{i=1}^{(x \dot{-} 1)+1} \mathrm{div}(x, i), \; \mathrm{chp}(x) = sg(|\mathrm{nd}(x) - 2|)$$

$$\pi(x) = \sum_{i=0}^{x} \overline{sg}(\mathrm{chp}(i)).$$

$$\mathrm{p}(x) = \mu_y(|\pi(y) - (x + 1)| = 0)$$

$$\mathrm{ex}(x, y) = \mu_u(sg(y) \cdot \mathrm{div}(y, (\mathrm{p}(x))^{u+1}) = 0)$$

Treba vediet F2, F4 a F8, teda x*y, sg a x/y
pri x/y nie je potrebne robit (x-1), horny limit moze byt cokolvek vacsie rovne (x-1)

chyba dodatok 29-49 https://matfyz.koniky.xyz/files/archive/3Z/vypoc/data/dodatok.pdf
chyba 02_dolneOdhady https://matfyz.koniky.xyz/files/archive/3Z/vypoc/data/02_dolneOdhady.pdf

preco je pri speedup theorem /log - kvoli transformacii na menej pasok
preco je K_0 take komplikovane - tazko z nularnej na unarnu funkciu
preco sa pri registrovych strojoch nepouziva binarna abeceda - este to nie je prepisane
preco je pocitanie num(1v_0^R) take komplikovane - lebo sa chcelo tymto sposobom poukazat na hodnotu toho cisla, da sa to jednoduchsie

preco je pri x/y pocitanie do (x-1)+1
preco je pri minimalizacii s ohranicenim to sg vonku a nie vnutri
preco pref ked je to konstantne vela

lema
veta
definicia
tvrdenie