Složitost I

doc. RNDr. Ondřej Čepek, Ph.D.

Obsah

přepsal Petr Hošek v ak. roce 2008/2009

1 Složitost

Jak porovnávat algoritmy?

- časová složitost algoritmu
- prostorová složitost algoritmu

Obě závisí na "velikosti" vstupních dat.

Jak měřit velikost vstupních dat? Počet bitů nutných k zapsání vstupních dat.

Definice (Časová složitost)**.** Funkce $f(|D|)$ udávající počet kroků algoritmu v závislosti na velikosti vstupních dat $|D|$.

Poznámka 1.1. Není podstatný přesný tvar funkce f (multiplikativní a aditivní konstanty), ale pouze to, do jaké "třídy" funkce f patří (lineární, kvadratická, exponenciální,...).

Co je to krok algoritmu? Operace daného abstraktního stroje (Turingův stroj, stroj RAM).

Poznámka 1.2. Krok algoritmu je operace proveditelná v konstantním (tj. na velikosti dat nezávislém) čase.

- aritmetické operace (sčítání, odčítání, násobení, dělení)
- porovnání dvou hodnot (typicky čísel)
- přiřazení (pouze pro jednoduché datové typy)

Toto zjednodušení nevadí při porovnávání algoritmů, ale může vést k chybě při zařazování algoritmů do tříd složitosti.

Proč měřit časovou složitost algoritmů? Za čas t na současném hardware umím zpracovat data velikosti $|D|$, odpovídající počet kroků je $f(|D|)$. Na q-krát rychlejším hardware $qf(|D|)$ kroků za čas t, odpovídající velikost dat je $f^{-1}(qf(|D|))$.

1.1 Asymptotická (časová) složitost

Zkoumá "chování" algoritmu na "velkých" datech, tj. nebere v úvahu multiplikativní a aditivní konstanty, pouze zařazuje algoritmy do "kategorií" podle jejich skutečné časové složitosti

(1) $f(n)$ je asymptoticky menší nebo rovno $g(n)$, značíme $f(n) \in O(g(n))$, pokud

$$\exists c > 0 \exists n_0 > 0 \forall n \geq n_0 : 0 \leq f(n) \leq c \cdot g(n)$$

(2) $f(n)$ je asymptoticky větší nebo rovno $g(n)$, značíme $f(n) \in \Omega(g(n))$, pokud

$$\exists c > 0 \exists n_0 > 0 \forall n \geq n_0 : 0 \leq c \cdot g(n) \leq f(n)$$

(3) $f(n)$ je asymptoticky stejné jako $g(n)$, značíme $f(n) \in \Theta(g(n))$, pokud

$$\exists c > 0 \exists d > 0 \exists n_0 > 0 \forall n \geq n_0 : 0 \leq c \cdot g(n) \leq f(n) \leq d \cdot g(n)$$

(4) $f(n)$ je asymptoticky ostře menší nebo rovno $g(n)$, značíme $f(n) \in o(g(n))$, pokud

$$\forall c > 0 \exists n_0 > 0 \exists n \geq n_0 : 0 \leq f(n) \leq c \cdot g(n)$$

(5) $f(n)$ je asymptoticky ostře větší nebo rovno $g(n)$, značíme $f(n) \in \omega(g(n))$, pokud

$$\forall c > 0 \exists n_0 > 0 \exists n \geq n_0 : 0 \leq c \cdot g(n) \leq f(n)$$

2 Algoritmy typu "Rozděl a panuj (Divide et impera)"

Metoda pro návrh algoritmů (ne dělení programu na samostatné celky). Algoritmus typu "Rozděl a panuj" má typicky 3 kroky

rozděl úlohu na několik podúloh stejného typu ale menšího rozsahu

vyřeš podúlohy, a to:

(1) rekurzivně dalším dělením pro podúlohy dostatečně velkého rozsahu

(2) přímo pro podúlohy malého rozsahu (často triviální)

sjednoť řešení podúloh do řešení původní úlohy

Příklad. Třídění sléváním (MergeSort).

2.0.1 Analýza časové složitosti

$T(n)$ doba zpracování úlohy velikosti n (předpokladáme, pokud $n < k$ tak $T(n) = \Theta(1)$)

$D(n)$ doba na rozdělení úlohy velikosti n na a podúloh stejné velikosti n/c

$S(n)$ doba na sjednocení řešení podúloh do řešení původní úlohy velikosti n.

Dostáváme rekurentní rovnici

$$\begin{array}{lcll} T(n) & = & D(n) + aT(n/c) + S(n) & \text{pro } n \geq k \\ T(n) & = & \Theta(1) & \text{pro } n < k. \end{array}$$

2.1 Metody řešení rekurentních rovnic

(1) substituční metoda

(2) Master theorem (řešení pomocí "kuchařky")

V obou případech používáme následující zjednodušení:

- předpoklad $T(n) = \Theta(1)$ pro dostatečně malá n nepíšeme explicitně do rovnice
- zanedbáváme celočíselnost, tj. např. píšeme $n/2$ místo $\lceil n/2 \rceil$ nebo $\lfloor n/2 \rfloor$
- řešení nás většinou zajímá pouze asymptoticky (nehledíme na konkrétní hodnoty konstant), asymptotická notace používána už v zápisu rekurentní rovnice

Příklad. Složitost algoritmu MergeSort je $T(n) = 2T(n/2) + \Theta(n)$

2.1.1 Substituční metoda

Uhodnout asymptoticky správné řešení a indukcí ověřit správnost odhadu (zvlášť pro dolní a horní odhad).

Příklad. Opět algoritmus MergeSort.

2.1.2 Master theorem

Definice (Jednoduchá verze)**.** Nechť $a \geq 1$, $c > 1$, $d \geq 0$ jsou reálná čísla a nechť $T : \mathbb{M} \to \mathbb{N}$ je neklesající funkce taková, že pro všechna n ve tvaru ck (kde $k \in \mathbb{N}$) platí

$$T(n) = aT(n/c) + F(n)$$

kde pro funkci $F : \mathbb{N} \to \mathbb{N}$ platí $F(n) = \Theta(n^d)$. Označme $z = \log_c a$. Potom

(1) je-li $z < d$, potom platí $T(n) \in \Theta(n^d)$,

(2) je-li $z = d$, potom platí $T(n) \in \Theta(n^d \log n) = \Theta(n^z \log n)$,

(3) je-li $z > d$, potom platí $T(n) \in \Theta(n^z)$.

Definice (Obecná verze)**.** Nechť $0 < a_i < 1$ (pro $1 \leq i \leq k$) a $d \geq 0$ jsou reálná čísla a nechť $T : \mathbb{N} \to \mathbb{N}$ splňuje rekurentní vztah $T(n) = T(a_1 n) + \ldots + T(a_k n) + F(n)$, kde pro $F : \mathbb{N} \to \mathbb{N}$ platí $F(n) = \Theta(n^d)$. Dále nechť je číslo z řešením rovnice $a_1^x + \ldots + a_k^x = 1$. Potom

(1) je-li $z < d$ (neboli $a_1^d + \ldots + a_k^d < 1$), potom platí $T(n) \in \Theta(n^d)$,

(2) je-li $z = d$ (neboli $a_1^d + \ldots + a_k^d = 1$), potom platí $T(n) \in \Theta(n^d \log n) = \Theta(n^z \log n)$,

(3) je-li $z > d$ (neboli $a_1^d + \ldots + a_k^d > 1$), potom platí $T(n) \in \Theta(n^z)$.

Poznámka 2.1. Jednoduchá verze je zjednoduššením obecné verze, totiž $\frac{1}{c}^x + \ldots + \frac{1}{c}^x = 1$ je rovno $a\frac{1}{c}^x = 1$, odtud $a = c^x$ a $x = \log_c a$.

Příklad. Opět algoritmus MergeSort.

2.2 Násobení čtvercových matic

Vstup jsou matice A a B řádu $n \times n$, výstupem potom matice $C = A \otimes B$ (také řádu $n \times n$).

2.2.1 Klasický algoritmus

```
   MULTIPLY(A, B)
for i ← 1 to n do
   for j ← 1 to n do
      C_{i,j} ← 0
      for k ← 1 to n do
         C_{i,j} ← C_{i,j} + A_{i,k} · B_{k,j}
      end for
   end for
end for
return C
```

Pozorování 2.2. *Časová složitost algoritmu je $T(n) = \Theta(n^3)$ (n^2 skalárních součinů délky n).*

Nyní předpokládejme že n je mocnina čísla 2 ($n = 2^k$), což umožňuje opakované dělení matice na 4 matice poloviční řádu až do matic řádu 1×1 a zkusme "rozděl a panuj" (předpoklad $n = 2^k$ později odstraníme).

$$A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \qquad B = \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix} \qquad C = \begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix}$$

$$\begin{aligned} C_{11} &= A_{11} \otimes B_{11} \oplus A_{12} \otimes B_{21} \\ C_{12} &= A_{11} \otimes B_{12} \oplus A_{12} \otimes B_{22} \\ C_{21} &= A_{21} \otimes B_{11} \oplus A_{22} \otimes B_{21} \\ C_{22} &= A_{21} \otimes B_{12} \oplus A_{22} \otimes B_{22} \end{aligned}$$

Poznámka 2.3. Každý skalární součin je "roztržen" na dvě poloviny a "dokončen" maticovým sčítáním. Počet maticových operací na maticích řádu $n/2$ je 8 násobení $\otimes$ a 4 sčítání $\oplus$. Počet sčítání (reálných čísel) v maticovém sčítání je $4(n/2)^2 = n^2$.

Pozorování 2.4. *Časová složitost algoritmu je* $T(n) = 8T(n/2) + \Theta(n^2)$*, Master theorem* $a = 8$*,* $c = 2$*,* $\log_c a = 3$*,* $d = 2$*, tedy* $T(n) = \Theta(n^3)$ *(asymptoticky stejné jako klasický algoritmus).*

Poznámka 2.5. Ke snížení složitosti je potřeba snížit $a = 8$ a zachovat nebo jen mírně zvýšit $d = 2$.

2.2.2 Strassenův algoritmus (1969)

Používá pouze 7 násobení submatic řádu $n/2$ (místo původních 8)

$$\begin{aligned}
M_1 &= (A_{12} \ominus A_{22}) \otimes (B_{21} \oplus B_{22}) \\
M_2 &= (A_{11} \oplus A_{22}) \otimes (B_{11} \oplus B_{22}) \\
M_3 &= (A_{11} \ominus A_{21}) \otimes (B_{11} \oplus B_{12}) \\
M_4 &= (A_{11} \ominus A_{12}) \otimes B_{22} \\
M_5 &= A_{11} \otimes (B_{12} \ominus B_{22}) \\
M_6 &= A_{22} \otimes (B_{21} \ominus B_{11}) \\
M_7 &= (A_{21} \oplus A_{22}) \otimes B_{11}
\end{aligned}$$

Poznámka 2.6. Počet maticových operací řádu $n/2$ je 7 násobení $\otimes$ a 10 sčítání $\oplus$ a odčítání $\ominus$.

$$\begin{aligned}
C_{11} &= M_1 \oplus M_2 \ominus M_4 \oplus M_6 \\
C_{12} &= M_4 \oplus M_5 \\
C_{21} &= M_6 \oplus M_7 \\
C_{22} &= M_2 \ominus M_3 \oplus M_5 \ominus M_7
\end{aligned}$$

Poznámka 2.7. Počet maticových operací řádu $n/2$ je 8 sčítání $\oplus$ a odčítání $\ominus$.

Pozorování 2.8. *Časová složitost algoritmu je* $T(n) = 7T(n/2) + \Theta(n2)$*, Master theorem* $a = 7$*,* $c = 2$*,* $\log_c a = \log_2 7 = x$*,* $d = 2$*, tedy* $T(n) = \Theta(n^x) = \Theta(n^{2.81})$.

Tvrzení 2.9. *Matice řádu* n*, kde* $n \neq 2^k$ *doplníme nulami do řádu* m*, tak aby* $m = 2^k$ *a* $n \leq m \leq 2n$ *a pro složitost Strassenova algoritmu platí* $\Theta(m^{\log_2 7}) = \Theta(n^{\log_2 7})$

Důkaz. Platí $n^{\log_2 7} \leq m^{\log_2 7} \leq (2n)^{\log_2 7} = 2^{\log_2 7} n^{\log_2 7} = 7n^{\log_2 7}$. ■

2.3 Hledání k-tého z n prvků

Vstupem algoritmu je neuspořádaná posloupnost n (různých) čísel, výstupem pak k-té nejmenší číslo. Časovou složitost algoritmu budeme pro jednoduchost měřit počtem porovnání. Pro $k = 1$ (minimum) a $k = n$ (maximum) jde triviálně pomocí $n - 1$ porovnání. Ukážeme že pro $k = n/2$ (medián) je to stejně těžké jako pro obecné k. První nápadem je setřídit posloupnost, potom vybrat k-tý prvek, toto řešení má časovou složitost $\Omega(n \log n)$.

Jde to lépe? Zkusíme použít metodu "Rozděl a panuj", z posloupnosti vybereme prvek (pivot) a podle něj roztřídíme posloupnost na tři části a to na m prvků menších než pivot, vybraného pivota a $(n - m - 1)$ prvků větších než pivot. Na to je potřeba $n - 1$ porovnání:

- pokud $k > m + 1$ tak zahodíme $m + 1$ malých prvků a hledáme $(k - m - 1)$-ní prvek mezi $(n - m - 1)$ prvky většími než pivot
- pokud $k = m + 1$ tak pivot je hledaný prvek a končíme

– pokud $k < m + 1$ tak zahodíme $n - m$ velkých prvků a hledáme k-tý prvek mezi m prvky menšími než pivot.

Tohle ovšem může špatně dopadnout pokud nezajistíme "dobrý" výběr pivota.

2.3.1 Algoritmus (Blum et al. 1972)

MEDIAN$(a_1, \ldots, a_n)$
Rozděl posloupnost délky n na $\lceil n/5 \rceil$ pětic (poslední může být neúplná).
V každé pětici najdi její medián.
Rekurzivně najdi medián ze získané množiny $\lceil n/5 \rceil$ mediánů.
Použij medián mediánů jako pivot k rozdělení vstupní posloupnosti.
Pokud medián mediánů není hledaným prvkem, tak rekurzivně hledej v množině prvků menších než je on nebo v množině prvků větších než je on.

Jak "dobré" je rozdělení podle pivota nelezeného výše uvedeným algoritmem?

Pozorování 2.10. *Pokud množina prvků menších než pivot i množina prvků větších než pivot každá obsahuje alespoň* $3n/10$ *prvků, pak v pátém kroku iteruji s nejvýše* $7n/10$*prvků.*

Důsledek 2.11. *Nechť* $T(n) =$ *počet porovnání použitý k nalezení* k*-tého z* n *prvků v nehorším případě, pak platí*

$$T(n) = 7n/5 + T(n/5) + (n-1) + T(7n/10).$$

Tvrzení 2.12. *Platí*

$$T(n) = O(n).$$

Důkaz. Substituční metodou (klíčový fakt je $1/5 + 7/10 < 1$, což při dělení na trojice nevyjde). Pokud bychom posloupnost dělili na trojice, pak by platilo $T(n) = 3n/3 + T(n/3) + (n-1) + T(2n/3)$ a pomocí substituční metody bychom dostali $T(n) = n \log n$ (platí $1/3 + 2/3 = 1$). Pro trojice tedy algoritmus v lineárním čase nefunguje, pro pětice, sedmice, atd. již ano. ■

3 Hladové algoritmy

Hladový algoritmus většinou splňuje následující podmínky

– v každém kroku udělá *lokálně* optimální rozhodnutí (výběr hodnoty)

– nikdy nemění již udělaná rozhodnutí ("nebacktrackuje")

– pokud vždy zaručeně nalezne *globálně* optimální řešení tak jde o hladový optimalizační algoritmus, jinak (většinou) hovoříme o hladové heuristice.

Definice (Problém 1)**.** Je dán souvislý neorientovaný graf $G = (V, E)$ a funkce $d : E \to \mathbb{R}^+$ udávající délky hran. Najděte minimální kostru grafu G, tj. acyklický podgraf grafu G, který má $|V| - 1$ hran, a mezi všemi takovými podgrafy má minimálním součet délek hran.

Definice (Problém 2)**.** Je dána množina $S = \{1, 2, \ldots, n\}$ úkolů jednotkové délky. Ke každému úkolu i je dána požadovaná lhůta dokončení $d_i \in \mathbb{N}^+$ a pokuta $w_i \in \mathbb{N}^+$, kterou je úkol penalizován není-li hotov do své lhůty. Najděte rozvrh (permutaci) úkolů (od času 0 do času n), který minimalizuje celkovou pokutu za úkoly, které nejsou hotovy do jejich lhůty.

Definice (Problém 3)**.** Je dána množina $S = \{1, 2, \ldots, n\}$ úkolů. Ke každému úkolu i je dán čas $s_i \in \mathbb{N}^+$ jeho zahájení a čas $f_i \in \mathbb{N}^+$ jeho dokončení, kde $s_i \leq f_i$. Úkoly i a j jsou kompatibilní pokud se intervaly $[s_i, f_i)$ a $[s_j, f_j)$ nepřekrývají. Najděte množinu po dvou kompatibilních úkolů, která je co největší.

Navrhneme obecný hladový algoritmus řešící problémy 1 a 2 jako speciální případy a dávající návod, jak řešit problém 3.

3.1 Matroidy

Definice. Matroid je uspořádaná dvojice $M = (S, I)$ splňující následující tři podmínky

(1) S je konečná a neprázdná množina prvků matroidu M.

(2) I je neprázdná množina podmnožin množiny S (nezávislých podmnožin množiny S), která má následující *dědičnou vlastnost*: pokud $(B \in I)$ a $(A \subseteq B)$ tak $(A \in I)$.

(3) I má následující *výměnnou vlastnost*: pokud $(A \in I)$ a $(B \in I)$ takové, že $(|A| < |B|)$ tak platí $(\exists x \in B \setminus A : A \cup \{x\} \in I)$.

3.1.1 Maticový matroid

Definice. Nechť T je reálná matice řádu $n \times n$. Pak T definuje matroid $M_T = (S_T, I_T)$ kde S_T je množina sloupců matice T a $A \subseteq S_T$ je nezávislá (tj. $A \in I_T$) pokud A sestává z lineárně nezávislých vektorů.

Věta 3.1. *$M_T = (S_T, I_T)$ je matroid.*

Důkaz. Dědičná vlastnost vyplývá z vlastností lineární nezávislosti, výměnná vlastnost pak platí díky větě o výměně. ■

3.1.2 Grafový matroid

Definice. Nechť $G = (V, E)$ je neorientovaný graf. Pak G definuje matroid $M_G = (S_G, I_G)$ kde $S_G = E$ a $A \subseteq S_G$ je nezávislá (tj. $A \in I_G$) pokud A neobsahuje cyklus (tj. pokud A tvoří les).

Lemma 3.2. *Nechť $G = (V, E)$ je neorientovaný graf a E' množina hran neobsahující cyklus. Pak $G' = (V, E')$ sestává z $|V| - |E'|$ stromů.*

Důkaz. Dokážeme matematickou indukcí podle počtu hran. Pro $|E'| = 0$ graf obsahuje $|V|$ izolovaných vrcholů, tedy stromů. Nechť nyní lemma platí pro $0, 1, \ldots, |E'|-1$, dokážeme jeho platnost pro $|E'|$. Odebereme hranu e, víme že $E' \setminus \{e\}$ sestává z $|V|-|E'|+1$ (dle indukčního předpokladu), po vrácení e klesne počet stromů o 1. ■

Věta 3.3. *$M_G = (S_G, I_G)$ je matroid.*

Důkaz. Dědičná vlastnost je triviální, dokážeme výměnnou vlastnost. Nechť $A, B \in I$, $|A| < |B|$, les A sestává z $|V| - |A|$ stromů, les B sestává z $|V| - |B|$ stromů a platí $|V| - |A| > |V| - |B|$ (dle lemma 3.2). Existuje strom T v B jehož vrcholy patří do (alespoň) dvou různých stromů A. Existuje hrana $e = (x, y) \in T \subseteq B$ taková, že x a y leží v různých stromech lesa A, $e \in B \setminus A$ a navíc $A \cup \{e\}$ je les. ■

Věta 3.4. *Všechny maximální nezávislé množiny v matroidu mají stejnou velikost.*

Důkaz. Dokážeme pro maticový matroid. Nechť $A, B \in I$ jsou maximální a navíc nechť platí $|A| < |B|$. Dle věty o výměnně $\exists x \in B \setminus A : A \cup \{x\} \in I$ což je spor s maximalitou A. ■

Definice. Matroid $M = (S, I)$ je vážený pokud je dána váhová funkce $w : S \to \mathbb{R}^+$ a její přirozené rozšíření na podmnožiny S je dáno předpisem $\forall A \subseteq S : w(A) = \sum_{x \in A} w(x)$.

3.1.3 Matroidový problém

Definice (Matroidový problém)**.** Pro daný vážený matroid M najděte nezávislou množinu v M s co největší váhou (taková množina se nazývá optimální množina matroidu M).

Poznámka 3.5. Protože jsou všechny váhy kladné, tak je optimální množina vždy (v inkluzi) maximální nezávislou množinou.

Pozorování 3.6. *Problém 1 je speciální případ matroidového problému.*

Důkaz. Vezměme $M_G = (S_G, I_G)$ a pro každé $e \in S_G$ definujme $w(e) = c - d(e)$, kde c je dostatečně velké číslo (libovolné $c > \max d(e)$ vyhovuje). Nyní platí, že optimální množina v M_G (vzhledem k w) je rovna minimální kostře v G (vzhledem k d). ■

Pozorování 3.7. *Problém 2 je speciální případ matroidového problému (toto není tak snadné jako předchozí případ).*

Definice. Nechť X je rozvrh všech úkolů. Úkol i nazveme

včasným v X pokud je ukončen v čase d_i nebo dříve

zpožděným v X pokud je ukončen až po čase d_i

Pozorování 3.8. *Při hledání optimálního rozvrhu stačí uvažovat pouze ty rozvrhy, ve kterých jsou všechny včasné úkoly rozvrženy před všemi zpožděnými.*

Pozorování 3.9. *Navíc lze předpokládat, že včasné úkoly jsou v rozvrhu uspořádány podle neklesajících lhůt dokončení (při shodě rozhoduje například číslo úkolu).*

Definice. Rozvrhy splňující podmínky 3.8 a 3.9 se nazývají kanonické.

Tvrzení 3.10. *Optimální rozvrh stačí hledat v množině kanonických rozvrhů. Navíc je každý kanonický rozvrh "jednoznačně" určen množinou svých včasných úkolů (až na permutaci zpožděných úkolů).*

Definice. Množina úkolů $A \subseteq S$ je nezávislá pokud existuje rozvrh X všech úkolů takový, že všechny úkoly v A jsou včasné v X. Označme I_S množinu všech nezávislých podmnožin množiny S.

Pozorování 3.11. *Množina včasných úkolů v libovolném kanonickém rozvrhu je nezávislá množina, což dává bijekci množiny včasných úkolů v kanonických rozvrzích a nezávislé množiny. A tudíž (díky výše uvedeným úvahám) platí minimalizace celkové pokuty rozvrhu je ekvivalentní minimalizaci součtu vah zpožděných úkolů v kanonickém rozvrhu, což je ekvivalentní maximalizaci součtu vah včasných úkolů v kanonickém rozvrhu, a to je ekvivalentní nalezení nezávislé množiny s co největší váhou (nalezení optimální množiny) což je přesně Matroidový problém.*

Definice. Definujme počet úkolů v C s lhůtou nejvýšš t jako $N_t(C) = |\{i \in C | d_i \leq t\}|$.

Lemma 3.12. *Množina C je nezávislá ($C \in I_S$) právě tehdy, když $\forall t \in \{0, \ldots, n\} : N_t(C) \leq t$.*

Důkaz.

$\Longrightarrow$ Dokážeme sporem, nechť $\exists t \in \{0, \ldots, n\} : N_t(C) > t$, potom C není nezávislá, protože do intervalu $[0, t]$ nelze rozvrhnout více než t úkolů se lhůtami menšími než t tak aby byly všechny včasné.

$\Longleftarrow$ Seřadíme-li úkoly v C podle lhůt, dostáváme rozvrh, kde jsou všechny úkoly v C včasné (nejvýšše jeden úkol s lhůtou 1, nejvýšše dva úkoly s lhůtou 2, atd.).

■

Věta 3.13. *$M = (S, I_S)$ je matroid.*

Důkaz. Dědičnost je triviální, dokážeme výměnnou vlastnost. Nechť $A, B \in I_S$ a $|A| < |B|$, označme $k = \max\{t|N_t(B) \leq N_t(A)\}$ (za předpokladu, že $\forall i : d_i \leq n$ lze udělat bez újmy na obecnosti). $N_n(B) = |B|$, $N_n(A) = |A|$, $|B| > |A|$ odkud $k < n$ a platí $\forall t \in \{k+1, \ldots, n\}$: $N_t(B) > N_t(A)$, tedy existuje úkol x s $d_x = k+1$ který je v $B \setminus A$. Nyní je $A \cup \{x\}$ nezávislá, stačí tedy dokázat, že $\forall t \in \{0, \ldots, n\} : N_t(A \cup \{x\}) \leq t$ (díky lemma 3.12). Platí $\forall t \in \{0, \ldots k\}$: $N_t(A \cup \{x\}) = N_t(A) \leq t$ a $\forall t \in \{k+1, \ldots, n\} : N_t(A_\cup\{x\}) = N_t(A) + 1 \leq N_t(B) \leq t$. ■

3.1.4 Hladový algoritmus pro Matroidový problém

Nechť je dán matroid $M = (S, I)$, kde $S = \{x_1, \ldots, x_n\}$, a váhová funkce $w : S \to \mathbb{R}^+$.

GREEDY(M, w)
$A \leftarrow \emptyset$
setřiď a přečísluj S sestupně (do nerostoucí posloupnosti) podle vah (tj. $w(x_1) = \max w(x_i)$)
for $i \leftarrow 1$ to n **do**
  if $(A \cup \{x\}) \in I$ **then**
    $A \leftarrow A \cup \{x\}$
  end if
end for
return A

Pozorování 3.14. *Časová složitost je $\Theta(n \log n)$ na setřídění, n-krát test na nezávislost (složitost záleží na konkrétním matroidu), celkem tedy $\Theta(n \log n + n \cdot f(n))$ kde $f(n)$ je složitost testu na nezávislost.*

Lemma 3.15. *Nechť $x \in S$ takový, že $\{x\} \notin I$. Potom neexistuje $A \in I$ taková, že $x \in A$.*

Důkaz. Vyplývá z dědičné vlastnosti. ■

Důsledek 3.16. *Prvky přeskočené před prvním výběrem nemohou být v žádné optimální množině.*

Lemma 3.17 (Přípustnost hladového výběru)**.** *Nechť je S setříděno podle vah do nerostoucí posloupnosti a nechť je x první prvek v S pro který $\{x\} \in I$. Potom existuje optimální množina $A \in I$ taková, že $x \in A$.*

Důkaz. Nechť B je optimální množina a $x \notin B$ (tj. $B \in I$). Pak $\forall y \in B : w(y) \leq w(x)$ a z $B \in I$ vyplývá $\{y\} \in I$. A zkonstruujume tak, že položíme $A = \{x\}$ a pomocí výměnné vlastnosti budeme zvětšovat A o prvky z B dokud nebude platit $|A| = |B|$. Potom $A = B \setminus \{y\} \cup \{x\}$ a $w(A) = w(B) + w(x) - w(y) \geq w(B)$, tedy A je optimální. ■

Důsledek 3.18. *První výběr "nezablokuje" cestu ke konstrukci optimální množiny, což znamená, že v další práci může algoritmus pátrat po optimální množině pouze mezi těmi množinami prvků, které obsahují prvek x.*

Lemma 3.19 (Existence optimální podstruktury)**.** *Nechť je x první prvek vybraný algoritmem* **GREEDY**(M, w)*. Pak je nalezení optimální množiny v M obsahující x ekvivalentní s nalezením optimální množiny v matroidu $M' = (S', I')$, kde $S' = \{y \in S | \{x, y\} \in I\}$ a $I' = \{B \subseteq S' \setminus \{x\} | (B \cup \{x\}) \in I\}$, váhová funkce w je stejná jako u M (omezená na S'). Matroid M' se nazývá kontrakce M prvkem x.*

Důkaz. Nejprve dokážeme, že M' je matroid. Z $A \in I'$ a $B \subseteq A$ vyplývá $B \in I'$. $A \cup \{x\} \in I$ a $B \cup \{x\} \subseteq A \cup \{x\}$ vyplývá $B \cup \{x\} \in I$ a tedy platí $B \in I'$ čímž jsme dokázali dědičnou vlastnost. Buď $A, B \in I'$ a $|A| < |B|$, z $A \cup \{x\} \in I$ a $B \cup \{x\} \in I$ vyplývá $\exists z \in (B \cup \{x\}) \setminus (A \cup \{x\})$: $A \cup \{x\} \cup \{z\} \in I$ odkud $A \cup \{z\} \in I'$ a $z \in B \setminus A$ čímž jsme dokázali výměnnou vlastnost matroidu M'.

Nyní dokážeme, že A je optimální v M obsahující x právě tehdy, když $A \setminus \{x\}$ je optimální v M'.

$\Longrightarrow$ Nechť A je optimální v M obsahující x a $B \in I'$ libovolná. Potom $B \cup \{x\} \in I$ odkud $w(A) \geq w(B \cup \{x\})$ a $w(A \setminus \{x\}) \geq w(B)$ a tedy $A \setminus \{x\}$ je optimální v M'.

$\Longleftarrow$ Nechť $A \setminus \{x\}$ je optimální v M' a $B \in I$ je libovolná obsahující x. Potom $B \setminus \{x\} \in I'$ odkud $w(A \setminus \{x\}) \geq w(B \setminus \{x\})$ a $w(A) \geq w(B)$ a $A \in I$. Tedy A je optimální v I mezi všemi nezávislými množinami obsahujícími x a tedy A je optimální v M.

■

Věta 3.20. **GREEDY**(M, w) *vrací optimální množinu matroidu* M.

Důkaz. Prvky přeskočené na počátku nemohou být v žádné optimální množině dle lemma 3.15. Po výběru prvního prvku x, lemma 3.17 zaručuje existenci optimální množiny obsahující x. Zbývající problém je převeden díky lemma 3.19 na problém stejného typu ale na menší množině prvků, iterujeme dokud existují neprázdné nezávislé množiny. ■

3.1.5 Hladový algoritmus pro Problém 3

Nechť $S = \{1, 2, \ldots, n\}$ je množina úkolů, $s_1, \ldots, s_n \in \mathbb{N}^+$ jsou časy zahájení a $f_1, \ldots, f_n \in \mathbb{N}^+$ jsou časy dokončení (kde $s_i \leq f_i$).

GREEDY-SELECT(s, f)
$A \leftarrow \emptyset$
$f_0 \leftarrow 0$
$j \leftarrow 0$
setřiď a přeznač pole s a f vzestupně podle časů dokončení f_i {tj. $f_1 = \min f_i$}
for $i \leftarrow 1$ to n **do**
  if $s_i \geq f_j$ **then**
    $A \leftarrow A \cup \{i\}$
    $j \leftarrow i$
  end if
end for
return A

Pozorování 3.21. *Časová složitost algoritmu je* $\Theta(n \log n)$ *na setřídění, zbytek je pak* $\Theta(n)$.

Lemma 3.22 (Přípustnost hladového výběru)**.** *Existuje optimální množina (tj. kompatibilní množina úkolů maximální kardinality) která obsahuje úkol s nejmenším* f_i.

Důkaz. Nechť B je optimální množina neobsahující úkol 1 ($f_1 \leq \min_{i \in B} f_i$) a nechť j je úkol v B s nejmenším f_j ($f_j = \min_{i \in B} f_i$). Potom $A = B \setminus \{j\} \cup \{1\}$ je kompatibilní a tedy A je optimální. ■

Lemma 3.23 (Existence optimální podstruktury)**.** *A je optimální množina pro S obsahující úkol 1 tehdy a jen tehdy, když $A \setminus \{1\}$ je optimální množina pro $S' = \{i | f_1 \leq s_i\}$.*

Důkaz. A je optimální pro S obsahující úkol 1 právě tehdy, když $A \setminus \{1\}$ je optimální pro $S' = \{i | f_1 \leq s_i\}$.

$\Longrightarrow$ Nechť A je optimální pro S obsahující úkol 1 a nechť $B \subseteq S'$ libovolná. Potom $B \cup \{1\}$ je kompatibilní odkud $|A| \geq |B \cup \{1\}|$ a $|A \setminus \{1\}| \geq |B|$. Tedy $A \setminus \{1\}$ je optimální pro S'.

$\Longleftarrow$ Nechť $A \setminus \{1\}$ je optimální pro S' a $B \in S$ je libovolná kompatibilní obsahující 1. Potom $B \setminus \{1\}$ je kompatibilní pro S' odkud $|B \setminus \{1\}| \leq |A \setminus \{1\}|$ a $|A| \geq |B|$. Tedy A je největší kompatibilní množina mezi všemi kompatibilními množinami obsahující úkol 1 a tedy A je optimální pro S.

■

Věta 3.24. GREEDY-SELECT(s, f) *vrací optimální množinu pro* S.

Věta 3.25. *Přímý důsledek lemmat 3.22 a 3.23.*

4 Grafové algoritmy

Značení. Graf $G = (V, E)$, V vrcholy, $|V| = n$, E hrany, $|E| = m$ (budeme používat stejné značení pro orientované i neorientovaný grafy).

Reprezentace grafů. Budeme používat pouze seznamy sousedů (velikost dat $\Theta(n + m)$).

4.1 Prohledávání grafů

4.1.1 Prohledávání do šířky (BFS – breadth first search)

```
   BFS(G, s)
for all u ∈ V do
   color(u) ← white
   d(u) ← ∞
   parent(u) ← nil
end for
color(s) ← grey
d(s) ← 0
parent(u) ← nil
Q ← {s}
while Q ≠ ∅ do
   u ← EXTRACT-FIRST(Q)
   S ← S ∪ u
   for all v ∈ V such that (u, v) ∈ E do
      if color(v) = white then
         color(v) ← grey
         d(v) ← d(u) + 1
         parent(v) ← u
         Q ← Q ∪ {v}
      end if
      color(u) ← black
   end for
end while
```

Poznámka 4.1. Prohledává graf po vrstvách podle vzdálenosti (měřeno počtem hran) od vrcholu s. Běží v čase $\Theta(n + m)$ a funguje i na orientovaném grafu (beze změny).

4.1.2 Prohledávání do hloubky (DFS – depth first search)

Poznámka 4.2. Neorientovaná verze, hlavní rozdíl proti BFS spočívá v tom, že aktivní (šedé) vrcholy nejsou ukládány do fronty ale do zásobníku, který je buď explicitně vytvářen algoritmem nebo implicitně rekurzivním voláním.

Plán. Ukážeme jak doplnit neorientovanou verzi DFS o další příkazy tak, že složitost zůstane lineární ($\Theta(n + m)$), ale algoritmus bude "plnit" podstatně složitější úkol.

```
  DFS(G)
for i ← 1 to n do
  color(i) ← white
  parent(i) ← 0
end for
for i ← 1 to n do
  if color(i) = white then
    VISIT(i)
  end if
end for

  VISIT(i)
color(i) ← grey
for all j such that (i, j) ∈ E do
  if color(j) = white then
    mark (i, j) as tree-edge
    parent(j) ← i
    VISIT(j)
  else if color(j) = grey and j ≠ parent(i) then
    mark (i, j) as return-edge
  end if
end for
color(i) ← black
```

4.2 Neorientované grafy

4.2.1 Testování 2-souvislosti neorientovaného grafu

Definice. Nechť $G = (V, E)$ je souvislý neorientovaný graf. Pak

(1) Vrchol $v \subseteq V$ je artikulace grafu G, pokud po odstranění v přestane být G souvislý.

(2) Graf G je 2-souvislý pokud nemá žádné artikulace.

(3) Množina hran $K \subseteq E$ tvoří 2-souvislou komponentu grafu G pokud je K v inkluzi maximální množina taková, že každé dvě hrany z K leží na prosté kružnici (tj. na kružnici bez opakování vrcholů).

Důsledek 4.3. *$v \in V$ je artikulace grafu G právě tehdy když $\exists x, y \in V, x \neq y$ takové, že každá cesta v G mezi x a y prochází vrcholem v.*

Důkaz.

$\Longrightarrow$ Vrchol v je artikulace, tedy po jeho odstranění se G rozpadne na alespoň 2 neprázdné komponenty. Vybereme vrcholy x a y v různých komponentách, všechny cesty z x do y tedy musí vést přes v.

$\Longleftarrow$ Existuje x a y takové že po odstranění vrcholu v jsou x a y v různých komponentách. Tedy vrchol v je artikulace.

■

Důsledek 4.4. *Graf G je 2-souvislý právě tehdy, když $\forall x, y \in V, x \neq y$ existují alespoň dvě vrcholově disjunktní cesty v G mezi x a y (odtud pojem 2-souvislost).*

Důkaz.

$\Longleftarrow$ Nechť v je libovolný vrchol, pak $\forall x, y$ v $G \setminus \{v\}$ existuje alespoň jedna cesta z x do y a tedy $G \setminus \{v\}$ je souvislý. Vrchol v tedy není artikulace a G je 2-souvislý.

$\Longrightarrow$ Nechť x, y jsou libovolné vrcholy a G je souvislý. Potom existuje cesta z x do y, důkaz indukcí dle délky d této cesty.

(1) Pro $d = 1$, nechť x není artikulace, pak existuje cesta z y do z nepoužívající hranu (x, y), tedy ani y není artikulace. Existuje cesta z x do z nepoužívající hranu (x, y) a tedy existuje cesta z x do y nepoužívající hranu (x, y). Existují tedy dvě vrcholově disjunktní cesty z x do y.

(2) Nechť platí pro $d = 1, 2, \ldots, d-1$ a z x do y vede cesta délky d. Dle indukčního předpokladu vedou z x do z dvě vrcholově disjunktní cesty A a B. z není artikulace, pak existuje cesta C z x do y v $G \setminus \{z\}$. Označme w poslední vrchol na cestě C, který je na cestě A nebo B. Bez újmy na obecnosti nechť je w na cestě A. První cesta z x do y vede z x do w po A, pak z w do y po C. Druhá cesta z x do y vede z x do z po B, pak po hraně (z, y).

■

Tvrzení 4.5. *Algoritmy na testování 2-souvislosti:*

(1) Triviální využití DFS, $\forall v \in V$ otestuje, zda je graf $G \setminus \{v\}$ souvislý (tj. 1-souvislý), což lze docílit pomocí n spuštění DFS v čase $\Theta(n(n+m)) = \Theta(nm)$ (protože G je souvislý a tedy $m \geq n-1$).

(2) Sofistikované využití DFS, v čase $\Theta(n+m) = \Theta(m)$ rozhodne o 2-souvislosti a zároveň označí všechny artikulace a 2-souvislé komponenty pomocí jediného spuštění "obohaceného" DFS

Pozorování 4.6. *Vrchol $i \in V$ je artikulací tehdy a jen tehdy když buď*

- *vrchol i není kořenem DFS stromu a má potomka j takového, že z podstromu s kořenem j nevede žádná zpětná hrana do nějakého předchůdce vrcholu i, nebo*
- *vrchol i je kořenem DFS stromu a má alespoň dva potomky.*

Jak artikulace najít? DFS doplníme tak, že budeme číslovat vrcholy podle pořadí jejich objevení indexy $d(i)$ a pro každý vrchol i navíc spočítáme funkci $\text{low}(i)$ definovanou následovně

Definice. Cestu z vrcholu i v DFS stromě nazveme *přípustnou* pokud vede z i "dolů" po libovolném (případně nulovém) počtu stromových hran a poté končí nejvýše jedním "skokem vzhůru" po zpětné hraně.

Definice.

$$\text{low}(i) = \min\{d(j) | \text{z } i \text{ do } j \text{ vede přípustná cesta}\}.$$

Definice.

$$\text{low}(i) = \min\{x, y, z\}$$

kde $x = d(i)$, $y = \min\{d(j) | \text{hrana } (i,j) \text{ je zpětná}\}$ a $z = \min\{\text{low}(j) | \text{hrana } (i,j) \text{ je stromová}\}$.

Jak spočítat $\text{low}(i)$**?**

(1) při objevení vrcholu i (přebarvení z bílý na šedý) inicializace $\text{low}(i) \leftarrow d(i)$

(2) při procházení seznamu sousedů vrcholu i pokud je soused j

bílý tak je rekurzivně navštíven (jako u obyčejného DFS) a hrana (i, j) se stává stromovou

šedý tak pokud

$j = parent(i)$ (tzn. že hrana (j, i) je stromová), tak se neděje nic

$j \neq parent(i)$ (tzn. že hrana (i, j) se stává zpětnou), tak proveď aktualizaci, tedy pokud platí $\text{low}(i) > d(j)$ tak dosaď $\text{low}(i) \leftarrow d(j)$

černý (tzn. že hrana (j, i) je zpětná) tak se neděje nic

(3) při backtrackingu do i (návrat po stromové hraně (i, j) z j do i) proveď aktualizaci, tedy pokud platí $\text{low}(i) > \text{low}(j)$ tak dosaď $\text{low}(i) \leftarrow \text{low}(j)$.

Jak se pozná, že vrchol i je artikulace?

(1) vrchol i není kořenem DFS stromu a má potomka j takového, že $d(i) \leq \text{low}(j)$ (což lze testovat při backtrackingu z vrcholu j do vrcholu i) nebo

(2) vrchol i je kořenem DFS stromu a má alespoň dva potomky (což budeme testovat pomocí jednoduchého triku)

4.3 Orientované grafy

4.3.1 Prohledávání do hloubky (DFS – depth first search)

Orientovaná verze, předpokládáme že graf je reprezentován pomocí seznamů sousedů.

4.3.2 Klasifikace hran pro DFS na orientovaném grafu

- (i, j) je stromová, j byl objeven z i, při prohlížení (i, j) je j *bílý*
- (i, j) je zpáteční, j je předchůdce i v DFS stromě, při prohlížení (i, j) je j *šedý*
- (i, j) je dopředná, i je předchůdce j v DFS stromě, při prohlížení (i, j) je j *černý* (ale ne přímý rodič) a navíc $d(i) < d(j)$
- (i, j) je příčná, nenastal ani jeden z předchozích tří případů, při prohlížení (i, j) je j *černý* a navíc $d(i) > d(j)$

4.3.3 Vlastnosti DFS

(1) Stromové hrany tvoří orientovaný les (DFS les = množina DFS stromů).

(2) Vrchol j je následníkem vrcholu i v DFS stromě, právě tehdy když v čase $d(i)$ existovala z i do j cesta sestávající výlučně z bílých vrcholů.

(3) Intervaly $[d(i), f(i)]$ tvoří "dobré uzávorkování" tj. pro každé $i \neq j$ platí

- buď $[d(j), f(j)] \cap [d(i), f(i)] = \emptyset$
- nebo $[d(i), f(i)] \subset [d(j), f(j)]$ a i je následníkem j v DFS stromě
- nebo $[d(j), f(j)] \subset [d(i), f(i)]$ a j je následníkem i v DFS stromě

(4) Vrchol j je následníkem i v DFS stromě právě tehdy, když $[d(j), f(j)] \subseteq [d(i), f(i)]$.

(5) Hrana (i, j) je příčná právě tehdy, když $f(j) < d(i)$ tj. všechny příčné hrany jdou "zprava doleva" (kreslíme-li DFS stromy a jejich podstromy "zleva doprava")

(6) Složitost je stále lineární $\Theta(n + m)$

```
   DFS-CON(G)
time ← 0
S ← ∅
for i ← 1 to n do
   color(i) ← white
   parent(i) ← 0
   d(i) ← 0
   root(i) ← false
   art(i) ← false
end for
for i ← 1 to n do
   if color(i) = white then
      root(i) ← true
      VISIT-CON(i)
   end if
end for

   VISIT-CON(G)
time ← time + 1
d(i) ← time
low(i) ← time
color(i) ← grey
for all j such that (i, j) ∈ E do
   if color(j) = white then
      mark (i, j) as tree-edge and put it in S
      parent(j) ← i
      VISIT-CON(j)
      if low(i) > low(j) then
         low(i) ← low(j)
      end if
      if d(i) ≤ low(j) then
         if ¬parent(i) then
            art(i) ← true
         end if
         remove edges from top of S to (i, j) including
      end if
      if parent(i) then
         parent(i) ← false
      end if
   else if color(j) = grey and j ≠ parent(i) then
      mark (i, j) as return-edge and put it in S
      if low(i) > d(j) then
         low(i) ← d(j)
      end if
   end if
end for
color(i) ← black
```

```
   DFS-ORIENTED(G)
time ← 0
for i ← 1 to n do
   color(i) ← white
end for
for i ← 1 to n do
   if color(i) = white then
      VISIT-ORIENTED(i)
   end if
end for

   VISIT-ORIENTED(i)
color(i) ← grey
time ← time + 1
d(i) ← time
for all j such that (i, j) ∈ E do
   if color(j) = white then
      VISIT-ORIENTED(j)
      mark (i, j) as tree-edge
   else if color(j) = grey then
      notify cycle
      mark (i, j) as return-edge
   else if d(i) < d(j) then
      mark (i, j) as forward-edge
   else
      mark (i, j) as traverse-edge
   end if
end for
color(i) ← black
time ← time + 1
f(i) ← time
```

4.3.4 Topologické číslování vrcholů orientovaného grafu

Definice. Funkce $t : V \to \{1, 2, \ldots, n\}$ je *topologickým očíslováním* množiny V pokud pro každou hranu $(i, j) \in E$ platí $t(i) < t(j)$.

Pozorování 4.7. *Topologické očíslování existuje pouze pro acyklické grafy.*

Tvrzení 4.8. *Algoritmus pro nalezení topologického očíslování vrcholů orientovaného grafu je mírnou modifikací DFS, běží v čase* $\Theta(n + m)$.

Lemma 4.9. *Graf G obsahuje cyklus právě tehdy, když* **DFS**(G) *najde zpětnou hranu.*

Věta 4.10. *Očíslování vrcholů acyklického grafu G podle klesajících časů jejich opuštění (časy $f(i)$) je topologické.*

Důkaz. Stačí dokázat $\forall(i, j) \in E : f(i) > f(j)$. Pokud barva j při zkoumání hrany (i, j) v DFS je

- bílá, pak j je objeven z i a stane se následníkem i, tedy $[d(j), f(j)] \subset [d(i), f(i)]$ a tedy $f(i) > f(j)$
- šedá, pak hrana (i, j) je zpětná a tedy v G je cyklus což je spor
- černá, pak $f(i) > f(j)$.

■

4.3.5 Silně souvislé komponenty orientovaného grafu

Definice. Nechť $G = (V, E)$ je orientovaný graf. Množina vrcholů $K \subseteq V$ se nazývá *silně souvislá komponenta* grafu G pokud

(1) Pro každou dvojici vrcholů $i, j \in K$ takových, že $i \neq j$ existuje v grafu G orientovaná cesta z i do j a orientovaná cesta z j do i.

(2) Neexistuje množina vrcholů L která by byla ostrou nadmnožinou K a splňovala (1).

Definice. Nechť $G = (V, E)$ je orientovaný graf. Graf $G^T = (V, E^T)$, kde

$$(i, j) \in E^T \iff (j, i) \in E$$

se nazývá *transponovaný graf* ke grafu G.

SSK(G)
Vstup: orientovaný graf $G = (V, E)$ zadaný pomocí seznamů sousedů
Fáze 1: **DFS**(G) doplněné o vytvoření spojového seznamu vrcholů podle klesajících časů jejich opuštění
Fáze 2: vytvoření transponovaného grafu G^T
Fáze 3: **DFS**(G^T) modifikované tak, že vrcholy jsou v hlavním cyklu zpracovávány v pořadí podle seznamu vytvořeného v 1. fázi (místo podle čísel vrcholů)
Výstup: DFS stromy z 3. fáze jsou silně souvislé komponenty grafu G

Pozorování 4.11. *Transponovaný graf lze zkonstruovat v čase* $\Theta(n + m)$ *a tím pádem celý algoritmus běží v čase* $\Theta(n + m)$.

Lemma 4.12. *Nechť $G = (V, E)$ je orientovaný graf a K je SSK v G. Po provedení* **DFS**(G) *platí:*

(1) množina K je podmnožinou vrcholů jediného DFS stromu

(2) v daném DFS stromě tvoří množina K podstrom

Důkaz.

(1) Mějme 2 DFS stromy a množina K je podmnožinou vrcholů obou těchto stromů. V tom případě musí existovat cesta mezi těmito stromy. Taková cesta může vést pouze jedním směrem a tedy K není SSK.

Mezi různými DFS stromy vedou pouze příčné hrany, ale pouze jedním směrem, tím pádem neexistuje cesta z x do y a tedy K není SSK.

(2) Mějme vrcholy v K, které nemají společného předka (v DFS stromě) v K

(a) K je podstrom DFS stromu, tedy lemma platí.

(b) Existují x a y v K takové, že v K již není žádný předchůdce x ani y (v DFS stromě). Cesta z x do y musí být celá uvnitř K, musí tedy použít příčnou hranu což je spor.

(c) K není podstrom, ale existuje společný předek v K. Potom $\exists x, y \in K$ takové, že x je předchůdce y a na cestě z x do y existuje $z \notin K$, což je spor s vlastnostmi SSK.

■

Věta 4.13. *Každý DFS strom z 3. fáze je silně souvislou komponentou grafu G.*

Důkaz. Mějme DFS stromy $S_1, \ldots, S_k$ vytvořené v první fázi, a $S'_1, \ldots, S'_l$ vytvořené v třetí fázi.

Nechť x je kořen S'_1 (první DFS-strom fáze 3), potom x je kořen S_k (poslední DFS strom fáze 1). Nechť $y \in S'$ je libovolné, pak existuje cesta v G^T z x do y což implikuje existenci cesty v G z y do x. Tedy $y \in S_k$ a tedy existuje cesta v G z x do y a tedy x a y patří do stejné SSK.

Z toho plyne že $S'_1 \subseteq SSK$ a navíc vrcholy v $S'_2, \ldots, S'_l$ nejsou v G^T dostupné z x, tedy nejsou ve stejné SSK jako x což implikuje že S'_1 je SSK.

Seznam stromů z první fáze upravím odstraněním vrcholů z S'_1 (použijeme lemma 4.12), kořen S'_2 je opět kořenem posledního DFS stromu v první fázi a iteruji postup. ■

4.4 Rovinné grafy

Definice. Neorientovaný graf $G = (V, E)$ se nazývá rovinný pokud existuje jeho vnoření (nakreslení) do Eukleidovské roviny takové, že se žádné dvě hrany neprotínají.

Tvrzení 4.14. *Pro daný graf $G = (V, E)$ lze v čase $\Theta(n + m)$ rozhodnout, zda je rovinný a v kladném případě zkonstruovat jeho rovinné vnoření.*

Definice. Nechť je $G = (V, E)$ rovinný graf s daným rovinným vnořením. Duální graf (resp. duální multigraf) $G^* = (V^*, E)$ je pak definován tak, že stěny G odpovídají vrcholům G^* a hrany G odpovídají hranám G^*.

Poznámka 4.15. Pokud existují v G dvě stěny s více než jednou společnou hranou, tak je G^* multigraf.

Tvrzení 4.16. *G^* je vždy souvislý. Pokud je i G souvislý, tak jsou G a G^{**} izomorfní, a navíc v takovém případě platí, že stěny G^* odpovídají vrcholům G. Pro daný vstupní graf G lze duální graf G^* zkonstruovat v čase $\Theta(n + m)$.*

Lemma 4.17. *Nechť je $G = (V, E)$ souvislý rovinný graf s daným rovinným vnořením, nechť je $G^* = (V^*, E)$ jeho duální graf a nechť $E' \subseteq E$. Pak podgraf (V, E') grafu G obsahuje cyklus tehdy a jen tehdy, když podgraf $(V^*, E \setminus E')$ grafu G^* není souvislý.*

Důkaz. Cyklus v G odpovídá řezu (kocyklu) v G^*. ■

Lemma 4.18. *Nechť je $G = (V, E)$, $G^* = (V^*, E)$ a $E' \subseteq E$ jsou dány jako v lemma 4.17. Pak (V, E') je kostra grafu G tehdy a jen tehdy, když $(V^*, E \setminus E')$ je kostra grafu G^*.*

Důkaz. Nechť (V, E) je kostra G, tedy (V, E) je souvislý a neobsahuje cyklus. Dle předchozího lemma $(V^*, E \setminus E')$ neobsahuje cyklus a je souvislý, tedy $(V^*, E \setminus E')$ je kostra $G*$. ■

Definice. Rovinný graf $G = (V, E)$ se nazývá *triangulovaný* pokud je každá jeho stěna trojúhelník, tj. každá jeho stěna je omezena právě třemi hranami. Graf $G' = (V, E')$ je triangulací grafu $G = (V, E)$ pokud je G' triangulovaný a $E \subseteq E'$.

Pozorování 4.19. *Pokud je graf G triangulovaný, tak je jeho duální graf G^* 3-regulární. Navíc lze pro každý rovinný graf G daný (nějakým) jeho rovinným vnořením zkonstruovat (nějakou) jeho triangulaci v čase $\Theta(n + m)$.*

4.5 Věta o planárním separátoru

Věta 4.20. *Nechť je $G = (V, E)$ neorientovanný rovinný graf. Pak existuje rozklad V na disjunktní množiny A, B a S takové, že*

(1) $A \cup B \cup S = V$ *(každý vrchol je v některé části rozkladu)*

(2) $(A \times B) \cap E = \emptyset$ *(množina S od sebe separuje množiny A a B)*

(3) $|A| \leq \frac{2}{3}n$

(4) $|B| \leq \frac{2}{3}n$

(5) $|S| \leq 4\sqrt{n}$

Toto rozdělení lze navíc zkonstruovat v čase $\Theta(n + m)$.

Důkaz. Bez újmy na obecnosti budeme předpokládat, že G je souvislý.

Zkonstruujeme rovinné vnoření grafu G (lze v linearním čase), vybereme libovolné $s \in V$ a provedeme **BFS**(G) z vrcholu s. Toto BFS rozdělí V do vrstev $L_0, \ldots, L_q$, kde $L_0 = s$. Dodefinujme $L_{q+1} = \emptyset$ a označme $|L_i| = z_i$, $i = 1, 2, \ldots, q + 1$.

Tvrzení 4.21. *Nechť $(x, y) \in E$. Pak buď $\exists i : \{x, y\} \subseteq L_i$ nebo $\exists i : x \in L_i$ a $y \in L_{i\pm1}$. Z toho plyne, že $\forall i : L_i$ je separátor oddělující $A = L_0 \cup \ldots \cup L_{i-1}$ a $B = L_{i+1} \cup \ldots \cup L_{q+1}$.*

Definice. Nechť t je index takový, že L_t obsahuje $n/2$-tý vrchol navštívený pomocí BFS.

Pozorování 4.22. *Pokud platí $z_t \leq 4\sqrt{n}$ tak jsme hotovi. Proto předpokládejme $z_t > 4\sqrt{n}$.*

Lemma 4.23. *Existují indexy $v < t$ a $w > t$ takové, že $z_v \leq \sqrt{n}$, $z_w \leq \sqrt{n}$ a $(w–v) \leq \sqrt{n}$.*

Značení. $C = L_0 \cup \ldots \cup L_{v-1}$, $D = L_{v+1} \cup \ldots \cup L_{w-1}$, $E = L_{w+1} \cup \ldots \cup L_q$ (víme $|C| < n/2$, $|E| < n/2$).

Pozorování 4.24. *Pokud platí $|D| \leq 2n/3$ tak jsme hotovi. Proto předpokládejme $|D| > 2n/3$.*

Důkaz. Nechť $|D| \leq 2n/3$, označme $S = L_v \cup L_w$. Pak A je největší z C, D, E a B jsou zbylé dvě množiny. ■

Nalezneme separátor S' velikosti $\leq 2\sqrt{n}$ rozdělující D v poměru 2 : 1 nebo vyrovnanějším. Potom definujeme

- $S = S' \cup L_v \cup L_w$
- $A =$ (větší z dvojice C, E) $\cup$ (menší kus D), horní odhad pro velikost A je $|A| \leq (n - |D|) + \frac{1}{2}|D| = n - \frac{1}{2}|D| \leq n - \frac{1}{2}\frac{2}{3}n = \frac{2}{3}n$
- $B =$ (menší z dvojice C, E) $\cup$ (větší kus D), horní odhad pro velikost B je $|B| \leq \frac{1}{2}(n - |D|) + \frac{2}{3}|D| = \frac{1}{2}n + \frac{1}{6}|D| \leq \frac{1}{2}n + \frac{1}{6}n = \frac{2}{3}n$

Tím budeme hotovi.

Z grafu G odstraníme vše kromě D a přidáme "nový" vrchol s, který spojíme hranou s každým vrcholem v L_{v+1} (graf zůstane rovinný). Získaný graf označíme $G' = (V', E')$.

Zkonstruujeme kostru T grafu G' jako:

(1) Pro každý vrchol $x \in L_{w-1}$ vyber jednu hranu jdoucí z x do L_{w-2} a dej ji do T.

(2) Opakuj předchozí krok pro vrcholy ve vrstvách $L_{w-2}, \ldots, L_{v+2}$.

(3) Dej do T všechny hrany z s do L_{v+1}.

Provedeme triangulaci grafu G', získaný graf označíme $G'' = (V'', E'')$ (platí $V'' = V'$) Vybereme cestu v T, která tvoří hledaný separátor S' (zbytek důkazu popisuje jak).

Lemma 4.25. *$\forall x, y \in V''$ cesta z x do y v kostře T má délku $\leq 2\sqrt{n}$ (T má průměr $\leq 2\sqrt{n}$).*

Definice. Hrany v $E'' \setminus T$ (čili ty co nejsou v kostře T) se nazývají *příčky* grafu G''. Označme $G^* = (V^*, E'')$ duální graf ke grafu G''.

Tvrzení 4.26. *$(V^*, E'' \setminus T)$ je kostra grafu G^* (tj. příčky z G'' tvoří kostru v G^*).*

Tvrzení 4.27. *$\forall e = (u, v) \in E'' \setminus T$ existuje v T právě jedna cesta z u do v, která spolu s e tvoří cyklus c_e.*

V dalčím postupu hledáme hranu e takovou, že cyklus c_e (což je cesta v T doplněná o e) tvoří požadovaný separátor, tj. cyklus c_e musí "obklíčit" něco mezi jednou třetinou a dvěmi třetinami všech vrcholů v D.

Vybereme list kostry $E'' \setminus T$ grafu G^*, označíme ho x^* a orientujeme všechny hrany kostry směrem od x^*.

Provedeme DFS na kostru $E'' \setminus T$ z vrcholu x^* a pro každou hranu $e \in E'' \setminus T$ induktivně spočítáme (v DFS stromě od listů vzhledem vzhůru, tj. při backtracku):

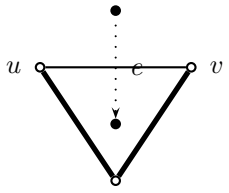
(1) U_e je počet vrcholů grafu G'' uvnitř cyklu c_e

(2) D_e je počet vrcholů na cyklu c_e (délka cyklu c_e)

(3) C_e je reprezentace cyklu c_e seznamem vrcholů

Tento výpočet závisí na typu hrany, musíme rozlišit 4 případy (nechť $e = (f, g) \in E'' \setminus T$):

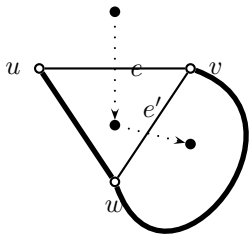
(1) Stěna g grafu G'' je omezena 2 hranami stromu T a 1 příčkou.



Vrchol g je list kostry $E'' \setminus T$, $U_e = 0, D_e = 3, C_e = [u, w, v]$.

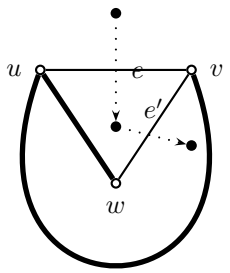
(2) Stěna g grafu G'' je omezena 1 hranou e stromu T a 2 příčkami

(a) Hrana stromu T je částí cyklu c_e.



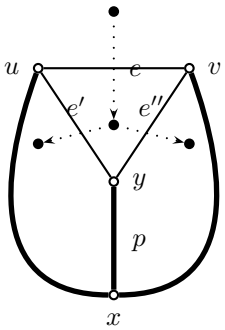
Hrana e má v DFS stromě jednoho následníka, $U_e = U_{e'}, D_e = D_{e'} + 1, C_e = [u] \circ C_{e'}$.

(b) Hrana stromu T není částí cyklu c_e



Hrana e má v DFS stromě jednoho následníka, $U_e = U_{e'} + 1$, $D_e = D_{e'} - 1$, $C_e = C_{e'} \setminus [w]$.

(3) Stěna g grafu G'' je omezena 3 příčkami.



Hrana e má v DFS stromě právě dva následníky, $U_e = U_{e'} + U_{e''} + |p| - 1$, $D_e = D_{e'} + D_{e''} - 2|p| + 1$, $C_e = C'_{e'} \circ [x] \circ C'_{e''}$ kde $C'_{e'}$ a $C'_{e''}$ vznikne z $C_{e'}$ a $C_{e''}$ odříznutím hrany p.

Lemma 4.28. *Existuje příčka e taková, že $U_e \leq 2n'/3$ a n'–$(U_e + D_e) \leq 2n'/3$ (kde $n' = |V''|$). Navíc e najdeme v průběhu práce DFS na kostře $E'' \setminus T$, tj. v čase $\Theta(n + m)$.*

Důkaz. Nechť e je první taková, že nastane $U_e + D_e \geq \frac{1}{3}n'$. Taková příčka existuje, protože v listech platí $U_e + D_e = 3$ a v kořeni $U_e + D_e = n'$. Pro takové e platí, že $n' - (U_e + D_e) \leq \frac{2}{3}n'$, čili zbývá ukázat, že $U_e \leq \frac{2}{3}n'$ (podle typu e)

(1) $U_e = 0$ a tedy platí

(2) (a) $U_e = U_{e'}$, $D_e = D_{e'} + 1$, tedy $U_{e'} + D_{e'} < \frac{1}{3}n'$ a odtud $U_e + D_e = U_{e'} + D_{e'} + 1 \leq \frac{2}{3}n'$ (pokud $n' \geq 3$)

(b) $U_e + D_e = U_{e'} + D_{e'}$ nemůže nastat

(3) $U_e + D_e = U_{e'} + U_{e''} + |p| - 1 + D_{e'} + D_{e''} - 2|p| + 1 = \underbrace{U_{e'} + D_{e'}}_{<\frac{1}{3}n'} + \underbrace{U_{e''} + D_{e''}}_{<\frac{1}{3}n'} - |p| \leq \frac{2}{3}n'$

■

Po nalezení hrany e zvolíme $S' = C_e \setminus \{s\}$, $X = U_e$, $Y = V'' \setminus \{U_e \cup C_e\}$, kde tentokrát chápeme U_e jako množinu vrcholů. Tím jsme hotovi. ■

Definice (Algoritmus pro konstrukci planárního separátoru)**.**

(1) Zkonstruuj vnoření $G = (V, E)$ do roviny (lineárním algoritmem *Hopcroft-Tarjan*).

(2) Proveď **BFS**(G) a rozděl V do vrstev $L_1, \ldots, L_{q+1}$.

(3) Najdi "prostřední" vrstvu L_t a pokud $z_t \leq 4\sqrt{n}$, tak jsme nalezli separátor ($S = L_t, A = L0 \cup \ldots \cup L_{t-1}, B = L_{t+1} \cup \ldots \cup L_{q+1}$).

(4) Najdi $v < t$ a $w > t$ takové, že $z_v \leq \sqrt{n}$, $z_w \leq \sqrt{n}$ a $(w–v) \leq \sqrt{n}$ a rozděl V na $C = L_0 \cup \ldots \cup L_{v-1}$, $D = L_{v+1} \cup \ldots \cup L_{w-1}$, $E = L_{w+1} \cup \ldots \cup L_q$. Pokud $|D| \leq 2n/3$, tak jsme nalezli separátor ($S = L_v \cup L_w$, $A =$ největší z $\{C, D, E\}$, $B =$ sjednocení zbývajících dvou z $\{C, D, E\}$).

(5) Zkonstruuj graf $G' = (V', E')$ přidáním vrcholu s k D a jeho kostru T o průměru $\leq 2\sqrt{n}$.

(6) Zkonstruuj graf $G'' = (V'', E'')$ triangulací grafu G'.

(7) Zkonstruuj graf $G^* = (V^*, E'')$ duální ke grafu G'' a jeho kostru $T^* = E'' \setminus T$.

(8) Proveď **DFS**(T^*) z listu kostry T^* a pro každou příčku $e \in T^*$ spočti U_e, D_e, C_e.

(9) Najdi příčku pro kterou platí $U_e \leq 2n'/3$ a $n'–(U_e + D_e) \leq 2n'/3$. Nechť $S' = C_e \setminus s$, $X = U_e$, $Y = V'' \setminus \{U_e \cup C_e\}$, kde U_e představuje množinu vrcholů. ($S = L_v \cup L_w \cup S'$, $A =$ větší z $\{X, Y\} \cup$ menší z $\{C, E\}$, $B =$ menší z $\{X, Y\} \cup$ větší z $\{C, E\}$)

Pozorování 4.29. *Každý krok trvá $O(n + m)$ a tedy celkově algoritmus trvá $\Theta(n + m)$.*

5 Amortizovaná složitost

Počítá amortizovanou cenu každé operace v nejhorším případě (v nejhorší možné posloupnosti operací). Dává realističtější odhad složitosti než klasický odhad složitosti každé operace nejhorším možným případem. Nejedná se o složitost v průměrném případě (přes nějakou distribuci vstupních dat).

Poznámka 5.1 (Amortizovaná analýza)**.** Budeme užívat tzv. *účetní metodu* (další metody jsou *agregační* a *potenciálová*), která funguje takto:

- od každé operace "vybereme" určitý "peněžní obnos" (amortizovanou cenu operace, která se může lišit podle typu operace), ze kterého "zaplatíme" za danou operaci
- jedna peněžní jednotka stačí na zaplacení konstantního množství práce
- pokud po zaplacení za operaci nějaké peníze zbudou, tak jsou uloženy na "účet"
- pokud pro zaplacení za operaci nějaké peníze chybí, tak je lze z "účtu" čerpat
- zůstatek na účtu musí být stále nezáporný (nic nelze provádět "na dluh")

Příklad (Inkrementace binárního čítače)**.** Inicializujeme k-bitový binarní čítač hodnotou $(0, \ldots, 0)$ a pak ho n krát inkrementujeme (předpokládáme $k \geq \log_2 n$). Cílem je spočítat *bitové operace (bit-flipy)* na jednu inkrementaci.

Klasicky počítaný nejhorší případ představuje n inkrementací a $\log_2 n$ bit-flipů na jednu incrementataci v nejhorším případě celkem $O(n \log_2 n)$ bit-flipů. Ukážeme, že amortizovaná cena jedné inkrementace je $2 = O(1)$ a tím pádem je celkový počet bit-flipů jenom $O(n)$.

Každá inkrementace změní právě jeden bit z 0 na 1, tedy jednu jednotku platí za tento flip a druhá jednotka je uloženo na účet u takto změněného bitu, potom každý bit jehož hodnota je 1 má na svém účtu jednu jednotku, které je později případně použito na flip z 1 na 0.

Příklad (Vkládání do dynamického pole)**.** Začneme s prázdným polem délky 1 a postupně do něj vložímen n prvků. Pokud je stávající pole plné, tak alokujeme pole dvojnásobné délky a stávající pole (a vkládaný prvek) do něj zkopírujeme. Cílem spočítat *operace s prvky pole* na jedno vložení.

Klasicky počítaný nejhorší případ představuje n vložení a $O(n)$ operací s prvky pole na jedno vložení v nejhorším případě celkově $O(n^2)$ operací s prvky pole . Ukážeme, že amortizovaná cena jednoho vložení je $3 = O(1)$ a tím pádem je celkový počet operací s prvky pole jenom $O(n)$.

Jedna jednotka na vlastní vložení, jedna na účet vloženého prvku a jedna na účet symetrického prvku v levé polovině pole. Před každou reorganizací pole má každý prvek právě jednu jednotku kterou "zaplatí" za kopírování do nového pole.

Definice (Dijkstrův Algoritmus)**.** Nechť $G = (V, E)$ je vážený orientovaný graf s váhami $w : E \to \mathbb{R}_0^+$ a nechť $s \in V$.

DIJKSTRA(G, w, s)
for all $v \in V$ **do**
  $d(v) \leftarrow \infty$
  $parent(v) \leftarrow$ **nil**
end for
$S \leftarrow \emptyset$
$Q \leftarrow V$
$d(s) \leftarrow 0$
while $Q \neq \emptyset$ **do**
  $u \leftarrow$ **EXTRACT-MIN**(Q)
  $S \leftarrow S \cup u$
  for all $v \in V$ such that $(u, v) \in E$ **do**
    if $d(v) > d(u) + w(u, v)$ **then**
      $d(v) \leftarrow d(u) + w(u, v)$
      $parent(v) \leftarrow u$
    end if
  end for
end while

Časová složitost algoritmu závisí na implementaci množiny Q.

- Q v poli:
  - naplnění pole $O(n)$,
  - **EXTRACT-MIN** $O(n)$,
  - **DECREASE-KEY** $O(1)$,

  celkem $O(n^2 + m) = O(n^2)$.
- Q v binární haldě:
  - vytvoření haldy $O(n \log_2 n)$,
  - **EXTRACT-MIN** $O(\log_2 n)$,
  - **DECREASE-KEY** $O(\log_2 n)$,

  celkem $O((n + m) \log n)$.
- Q ve Fibonacciho haldě:
  - vytvoření haldy $O(n)$,
  - **EXTRACT-MIN** $O(\log_2 n)$,
  - **DECREASE-KEY** $O(1)$.

  celkem $O(n \log n + m)$.

Příklad.

(1) Řídký graf kde $m = \Theta(n)$:
  - Q v poli $O(n^2)$,
  - Q v binární haldě $O(n \log n)$,
  - Q ve Fibonacciho haldě $O(n \log n)$.

(2) Hustý graf kde $m = \Theta(n^2)$:

- Q v poli $O(n^2)$,
- Q v binární haldě $O(n^2 \log n)$,
- Q ve Fibonacciho haldě $O(n^2)$.

(3) V grafu kde $m \in o(n^2)$ a $m \in \omega(n \log n)$ je Fibonacciho halda ostře lepší než pole tak binární halda.

5.1 Binomiální haldy

Definice. *Binomiální strom* B_i je rekursivně definován jako strom sestávající z kořene a jeho dětí $B_0, B_1, \ldots, B_{i-1}$. Strom B_0 sestává z jediného vrcholu. Každý binomiální strom má *vlastnost haldy*, tj. pro každou stromovou hranu platí klíč otce $\leq$ klíč syna.

Poznámka 5.2. B_i sestává ze dvou kopií B_{i-1}, ze kterých vznikne operací slévání.

Definice. *Řád uzlu* je počet jeho synů, *řád stromu* je řád jeho kořene. *Binomiální halda* je soubor binomiálních stromů, ve které žádné dva stromy nemají stejný řád, doplněný o ukazatel min ukazující na kořen s nejmenším klíčem.

Lemma 5.3. $|B_i| = 2^i$ *pro každé* i *a každý strom v haldě s* n *vrcholy má řád nejvýše* $\log_2 n$.

Důkaz. Platí $B_0 = 1$ a $B_i = 2B_{i-1}$. ■

Poznámka 5.4. Pro binomiální haldu mám ukazatel min, který ukazuje na nejmenší prvek haldy což je vrchol některého stromu B_i.

5.1.1 Pilná representace binomiální haldy

Kořeny stromů v haldě jsou dostupné z pole ukazatelů, kde i-tý ukazatel buď ukazuje na kořen stromu B_i nebo má hodnotu **nil**. Následující operace jsou na binomiální haldě podporovány:

MERGE(h, h') pracuje v čase $O(\log_2 n)$ (podobné jako binarní sčítání dvou čísel),

EXTRACT-MIN(h) volá $Merge$ a pracuje v čase $O(\log_2 n)$, vyhodí kořen na který ukazuje min, ze synů tohoto kořene vytvoří haldu a nastaví její min a na obě haldy zavolá operaci **MERGE**,

INSERT$(h, i) =$ **MERGE**$(h,$ **MAKEHEAP**$(i))$ pracuje amortizovaně v čase $O(1)$, stejně jako inkrementace binárního čítače v příkladu 5.

Tvrzení 5.5 (Invariant (platný po celou dobu existence haldy))**.** *Každý binomiální strom v haldě má na svém účtu (vedeném v kořeni stromu) 1 jednotku, které bylo zaplaceno operací, která tento strom vytvořila.*

5.1.2 Líná representace binomiální haldy

Kořeny stromů v haldě jsou svázány dvousměrným kruhovým seznamem. Seznam může dočasně obsahovat více stromů stejného řádu.

MERGE(h, h') nyní pracuje v $O(1)$

INSERT(h, i) znovu pracuje v $O(1)$ (tentokrát volá líný **MERGE**)

EXTRACT-MIN(h) volá **CONSOLIDATION** (do pilné representace) a pracuje v $O(\log_2 n)$

Operace **CONSOLIDATION** je implementována následovně

(1) Vytvoří pole (stejně jako u pilné reprezentace 5.1.1) (plné **nil**), složitost $O(\log_2 n)$.

(2) Postupně bere stromy ze spojového seznamu a zavěšuje je do pole, s tím že za stromy

   (a) jejichž kořeny zůstanou kořeny i po zpracování všech stromů zaplatí **CONSOLIDATION**

   (b) jejichž kořeny přestanou být kořeny zaplatí účet kořene (vyjmutí ze seznamu a následné slití)

   těchto je $O(\log_2 n)$.

(3) Z pole opět vytvořím spojový seznam a nastavím min, složitost $O(\log_2 n)$.

Operace **EXTRACT-MIN** funguje v čase $O(\log_2 n)$ díky faktu, že velikost každého binomiálního stromu je exponenciální vůči jeho řádu.

5.2 Fibonacciho haldy

Stromy jsou opět v dvousměrném kruhovém seznamu (jako v líné implementaci 5.1.2). Stromy nemusí být binomiální, ale slévání opět probíhá jen mezi stromy stejného řádu.

MERGE(h, h'), **INSERT**(h, i), **EXTRACT-MIN**(h) implementovány jako v líné implementaci 5.1.2. **DECREASE-KEY**(h, i, d) sníží klíč prvku (vrcholu) i v haldě h o množství d

- odřízne podstrom s kořenem i a zavede ho do seznamu jako samostaný strom
- od každého vrcholu x smí být odříznuti nejvýše dva z jeho synů – poté je sám x odříznut i se svým zbývajícím podstromem
- pracuje amortizovaně v $O(1)$ (přestože může spustit kaskádu řezů).

DECREASE-KEY bude mít amortizovanou cenu $4 = O(1)$

- 1 jednotka na práci (odříznutí uzlu kde snižuji klíč a zavedení do spojového seznamu)
- 1 jednotka na účet nového stromu
- 2 jednotky otci odříznutého uzlu.

Když uzel ztrácí druhého syna, tak má naspořeny 4 jednotky, které použije stejně jako je popsáno výše.

Lemma 5.6. *Nechť x je vrchol a $y_1, \ldots, y_m$ jeho synové v pořadí, ve kterém byli pod x sliti. Potom $\forall i \in \{1, \ldots, m\}$ je řád vrcholu y_i roven alespoň $i - 2$.*

Důkaz. V okamžiku, kdy vznikla hrana (x, y_i) byly x a y_i kořeny dvou stromů stejného řádu. Navíc, řád x byl v tom okamžiku $\geq i - 1$ (protože $y_1, \ldots, y_{i-1}$ již byly syny x). Od té doby mohl y_i ztratil nejvýšše jednoho syna (jinak by byl sám odříznut od x), tedy řád $y_i \geq i - 2$. ■

Značení. Nechť F_i je nejmenší možný (počtem vrcholů) strom ve Fibonacciho haldě.

Pozorování 5.7. *F_i se skládá z kopie F_{i-1} a kopie F_{i-2}.*

Značení. $f_i = |F_i|$

Věta 5.8. *Strom řádu i ve Fibonacciho haldě obsahuje alespoň φ^i vrcholů pro nějaké $\varphi > 1$.*

Důkaz. Chci aby $f_i \geq \varphi^i$. φ je kořen rovnice $x^2 = x + 1$, a platí $\varphi = \frac{1+\sqrt{5}}{2}$. Dokážeme indukcí podle i. Pro $i = 0$ platí $f_0 = 1 \geq \varphi^0 = 1$. Nechť nyní věta platí pro $i - 1$ a dokážeme pro i, platí $f_i = f_{i-1} + f_{i-2} \geq \varphi^{i-1} + \varphi^{i-2} = \varphi^{i-2}(\varphi + 1) = \varphi^{i-2} \cdot \varphi^2 = \varphi^i$. ■

6 Složitostní třídy

Postupně ukážeme, že následující 4 problémy (úlohy) jsou "stejně těžké":

Klika (KL) Je dán neorientovaný graf $G = (V, E)$. Najděte největší kliku (úplný podgraf maximální kardinality) v grafu G.

Splnitelnost (SAT) Je dána KNF $\mathcal{F}$ n Booleovských proměnných. Existuje pravdivostní ohodnocení proměnných které splňuje $\mathcal{F}$?

Obchodní cestující (TSP) Je dán ohodnocený neorientovaný graf $G = (V, E)$ s nezápornými váhami na hranách. Najděte v G Hamiltonovskou kružnici minimální celkové váhy.

Součet podmnožiny (SP) Je dána množina n přirozených čísel $a_1, \ldots, a_n$ a dále přirozené číslo b. Existuje $S \subseteq \{1, \ldots, n\}$ takové, že $\sum_{i \in S} a_i = b$?

Úloha pro daný vstup (instanci úlohy) je cílem získat výstup s danými vlastnostmi (např. najít kostru v grafu, cyklus v grafu, Hamiltonovskou kružnici, ...).

Optimalizační úloha úloha, kde je cílem získat optimální (zpravidla nejmenší / největší) výstup s danými vlastnostmi (např. min. kostra, nejkratší cyklus, klika, TSP, ...).

Rozhodovací problém úloha, jejímž výstupem je odpověď ano/ne (např. SAT, SP).

V dalším výkladu se omezíme pouze na rozhodovací problémy, což není příliš omezující podmínka, protože ke každé optimalizační úloze lze snadno "přiřadit" nejvýše stejně těžký (z hlediska řešitelnosti v polynomiálním čase) rozhodovací problém (a často i naopak, ale tento směr nebývá tak snadný).

6.0.1 Kódování vstupů

Nechť P je rozhodovací problém, předpokládáme, že každé zadání (instance) problému P je zakódována jako řetězec (slovo) v abecedě $\{0, 1\}$, tj. instance problému je slovo z abecedy $\{0, 1\}^*$. Kódy všech instancí problému P jsou reprezentovány v jazyce $L(P) \subseteq \{0, 1\}^*$, který se dělí na

- $L(P)_Y$ = kódy instancí s odpovědí ano
- $L(P)_N$ = kódy instancí s odpovědí ne

Rozhodovací problém P lze nyní formulovat takto – pro daný vstup $x \in L(P)$ rozhodni zda $x \in L(P)_Y$ nebo $x \notin L(P)_N$. Předpokládáme, že rozhodnout zda $x \in L(P)$ (tj. rozhodnout zda x kóduje nějakou instanci problému P nebo je to "nesmysl") lze v polynomiálním čase vzhledem k $|x|$ (tento test lze považovat za jakýsi "preprocessing").

6.0.2 Deterministický Turingův Stroj (DTS)

DTS je abstraktní stroj, který se skládá z:

- *řídící jednotky* která může být v konečném počtu stavů
- potenciálně nekonečné (jednostranné) *pracovní pásky* (která sestává z buňek, z nichž každá může obsahovat právě jeden symbol)
- *hlavy* pro čtení a zápis pracující na pásce.

Program pro DTS sestává z:

- konečné množiny Q stavů řídící jednotky, která obsahuje počáteční stav q_0 and koncové stavy q_y and q_n

– konečné množiny Γ páskových symbolů, která obsahuje množinu $\Sigma \subseteq \Gamma$ vstupních symbolů a prázdný symbol $\lambda \in \Gamma \setminus \Sigma$

– přechodové funkce $\delta : (Q \setminus \{q_y, q_n\}) \times \Gamma \to Q \times \Gamma \times \{\leftarrow, \circ, \rightarrow\}$.

DTS program začíná v počátečním stavu q_0 s hlavou na prvním buňce pracovní pásky a vstupním slovem $x \in \Sigma^*$ v prvních $|x|$ buňkách pracovní pásky (všechny ostatní buňky obsahují λ). Každý krok programu spočívá v jednom použití přechodové funkce. Program končí práci v okamžiku, když dosáhne koncového stavu. Program buď přijme x ve stavu q_y nebo odmítne x ve stavu q_n.

Definice. Jazyk $L(M)$ přijímaný DTS programem M sestává ze všech vstupních slov přijímaných strojem M. DTS program M řeší rozhodovací problém P pokud M vždy zastaví (pro každý vstup) a navíc platí $L(M) = L(P)_Y$.

Definice. Nechť M je DTS program, který vždy zastaví. Časová složitost programu M je dána funkcí $T_M(n) = \max\{t | \exists x \in \Sigma^*, |x| = n, M$ skončí na vstupu x po t krocích$\}$. Pokud navíc existuje polynom p takový, že $\forall n : T_M(n) \leq p(n)$, tak se M nazývá polynomiální DTS program.

Definice. Rozhodovací problém P je ve *třídě* **P** tehdy a jen tehdy, když existuje polynomiální DTS program M který řeší P, tj. takový, že $L(M) = L(P)_Y$.

6.0.3 Nedeterministický Turingův stroj (NTS)

Všechny definice stejné jako pro DTS kromě přechodové funkce δ, která je nahrazena tabulkou δ, která přiřazuje každé dvojici z množiny $(Q \setminus \{q_y, q_n\}) \times \Gamma$ množinu možných pokračování, tj. množinu trojic z $Q \times \Sigma \times \{\leftarrow, \circ, \rightarrow\}$.

Definice. NTS s programem M přijímá vstup $x \in \Sigma^*$ tehdy a jen tehdy když existuje přijímající výpočet programu M na vstupu x (výpočet končící v q_y). Jazyk $L(M)$ přijímaný NTS s programem M sestává ze všech vstupních slov přijímaných strojem M. Čas ve kterém M přijímá vstup x je definován jako počet kroků v nejkratším přijímacím výpočtu.

Definice. Časová složitost NTS M je dána funkcí $T_M(n)$, kde $T_M(n) = 1$ pokud M nepřijímá žádný vstup x s $|x| = n$ a $T_M(n) = \max\{t | \exists x \in \Sigma^*, |x| = n, M$ přijímá vstup x v t krocích$\}$ v ostatních případech. Pokud navíc existuje polynom p takový, že $\forall n : T_M(n) \leq p(n)$, tak se M nazývá polynomiální NTS program (na nepřijímaných vstupech nemusí, na rozdíl od DTS, výpočet M vůbec skončit).

Definice. Rozhodovací problém P je ve *třídě* **NP** tehdy a jen tehdy, když existuje polynomiální NTS program M takový, že $L(M) = L(P)_Y$, a ve *třídě* **co-NP** tehdy a jen tehdy, když existuje polynomiální NTS program M takový, že $L(M) = L(P)_N$.

Definice (Alternativní definice NTS)**.** Přidáme tzv. hádací pásku. Práce NTS pak sestává ze 2 fází

(1) Na hádací pásku je (nedeterministicky) zapsán řetězec $y \in \{1, \ldots, k\}^*$, kde k je maximální počet možných pokračování pro dvojici (stav, symbol) v tabulce δ.

(2) Pomocí y je výpočet na vstupu x změněn z nedeterministického na deterministický. Vstup x je přijat pokud existuje (certifikát) y kódující přijímající výpočet.

7 NP-těžkost a NP-úplnost

Turingovy stroje, o kterých jsem zatím mluvili jsou tzv. *akceptory* (vstup přijímají nebo nepřijímají). Kromě akceptorů lze definovat také tzv. *transducery*.

Definice. Turingův stroj (deterministický nebo nedeterministický) doplněný o výstupní pásku, na kterou lze pouze zapisovat, a jejíž hlava se pohybuje pouze vpravo, se nazývá transducer. Výstupem takového TS je obsah výstupní pásky v okamžiku, kdy TS zastaví. Ostatní pojmy (např. časová složitost) jsou definovány jako u akceptorů.

Definice. Funkce $f : \{0,1\}^* \to \{0,1\}^*$ je polynomiálně vyčíslitelná tehdy a jen tehdy, když existuje polynom p a DTS transducer A takový, že pro každý vstup $x \in \{0,1\}^*$ dává DTS transducer A výstup $f(x)$ po vykonání nejvýše $p(|x|)$ kroků.

Definice. Jazyk L je polynomiálně převoditelný na jazyk L' (značíme $L \propto L'$) tehdy a jen tehdy, když existuje polynomiálně vyčíslitelná funkce f taková, že

$$x \in \{0,1\}^* : (x \in L) \equiv (f(x) \in L').$$

Definice. Problém P je *NP-těžký* tehdy a jen tehdy, když $\forall Q \in \mathbf{NP} : L(Q)_Y \propto L(P)_Y$. Problém P je *NP-úplný* tehdy a jen tehdy, když je P *NP-těžký* a navíc $P \in \mathbf{NP}$.

Definice. Problém P je *co-NP-těžký* tehdy a jen tehdy, když $\forall Q \in \mathbf{co\text{-}NP} : L(Q)_N \propto L(P)_N$. Problém P je *co-NP-úplný* tehdy a jen tehdy, když je P *co-NP-těžký* a navíc $P \in \mathbf{co} - \mathbf{NP}$.

Definice. Pokud je Q NP-těžký a $L(Q)_Y \propto L(P)_Y$, pak také P je NP-těžký. Obdobně, pokud je Q co-NP-těžký a $L(Q)_N \propto L(P)_N$, pak také P je co-NP-těžký. Toto je standardní postup jak dokazovat NP-těžkost (resp. co-NP-těžkost), který ovšem vyžaduje existenci alespoň jednoho NP-těžkého (resp. co-NP-těžkého) problému.

Věta 7.1 (Cook-Levin (1971))**.** *Existuje NP-úplný problém (původní důkaz je udělán pro SAT).*

Definice (Kachlíkování ($KACHL$))**.** Zadáním je množina barev B, čtvercová síť S s obvodem obarveným barvami z B, množina K typů kachlíků, kde je každý typ definován svou horní, dolní, levou a pravou barvou. Otázkou je zda lze síť S vykachlíkovat pomocí kachlíků z množiny K (stejný typ lze použít libovolně krát, kachlíky ale nelze otáčet) tak, aby barvy kachlíků přilehlé k obvodu sítě souhlasily s barvami předepsanými tomto na obvodu sítě a každá dvojice barev na dotyku dvou kachlíků byla rovněž shodná?

Věta 7.2. *Kachlíkování je NP-úplné.*

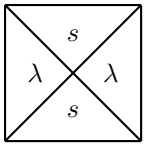
Důkaz.

(1) $KACHL$ je v NP, to je zřejmé.

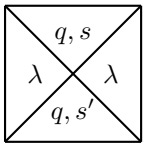
(2) $KACHL$ je NP-těžký, dokážeme z definice třídy NP. Nechť $R \in \mathbf{NP}$ libovolný a nechť M je NTS program rozpoznávající $L(R)_Y$ v čase shora omezeném polynomem p (tj. $\forall x \in (R)_Y$ skončí nejkratší přijímající výpočet stroje M na vstupu x po nejvýše $p(|x|)$ krocích). Navíc předpokládejme, že M přijímá "s prázdnou páskou" (po přechodu do stavu q_Y neskončí práci, ale vše na pásce přepíše prázdným symbolem, hlavu "zaparkuje" na začátku pásky a přejde do "nového" koncového stavu). Nechť je M definován množinou stavů Q, vstupní abecedou $\Sigma = \{0,1\}$, pracovní abecedou $\Gamma = \{0,1,\lambda\}$ (abecedy lze případně překódovat) a přechodovou tabulkou δ. Nechť zadání $x \in \{0,1\}^*$ problému R, kde $|x| = n$, je vstupem programu M. Zkonstruujeme zadání $f(x)$ problému KACHL takové, že

$$x \in L(R)_Y \iff f(x) \in L(KACHL)_Y.$$

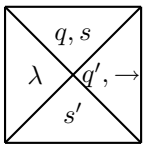
Buď $B = \{0,1,\lambda\} \cup Q \times \{0,1,\lambda\} \cup Q \times \{\leftarrow, \rightarrow\}$ (má konstatní velikosti). S má velikost $p(|x|) \times p(|x|)$ (zajímají nás tedy pouze polynomiálně dlouhé přijímající výpočty), horní okraj sítě má tvar $(q_0, x_1), x_2, x_3, \ldots, x_n, \lambda, \ldots, \lambda$, dolní okraj $(q_y, \lambda), \lambda, \ldots, \lambda$; levý a pravý okraj má tvar $\lambda, \ldots, \lambda$. Vstup má tvar $x = x_1 x_2 \ldots x_n$. Množina K obsahuje následující typy kachlíků

(a) 

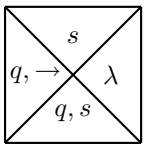
$\forall s \in \{0, 1, \lambda\} = \Gamma$

(b) 

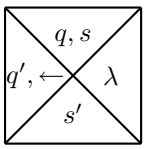
pokud $(q', s', \circ) \in \delta(q, s)$

(c) 

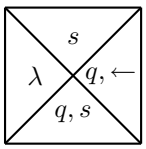
pokud $(q', s', \rightarrow) \in \delta(q, s)$

(d) 

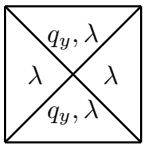
$\forall q \in Q\ \forall s \in \Gamma$

(e) 

pokud $(q', s', \leftarrow) \in \delta(q, s)$

(f) 

$\forall q \in Q\ \forall s \in \Gamma$

(g) 

Horní barvy kódují konfiguraci před provedením kroku stroje, dolní barvy pak konfiguraci po provedení kroku stroje. Symboly reprezentují obsah pásky, (q, s) určuje stav stroje a čtený symbol, s potom obsah pásky. Pokud mám přijímající výpočet, dokážu pomocí něj sestavit správné vykachlíkování. Pokud mám správné vykachlíkování, mohu z něj vytvořit přijímající výpočet.

■

Poznámka 7.3. Jak z NTS M udělat NTS M' přijímající s prázdnou páskou? Namísto q_Y má M' "nový stav", který spustí úklid

(1) hlava jede doprava na poslední navštívené políčko (prázdný symbol má dvojníka, hledám původní prázdný symbol který určuje nenavštívenou část pásky)

(2) hlava jede vlevo a vše přepisuje na prázdný symbol.

7.0.4 Splnitelnost (SAT)

Definice. KNF $\mathcal{F}$ na n Booleovských proměnných. Existuje pravdivostní ohodnocení proměnných, které splňuje formuli $\mathcal{F}$?

Věta 7.4. *SAT je NP-úplný.*

Důkaz. $SAT \in \mathbf{NP}$ a dokážeme, že $KACHL \propto SAT$.

Mějme B, S, K a potřebujeme získat KNF $\mathcal{F}$. Nechť $|S| = n \times n$. Definujme proměnné jako

$$x_{ijk} = \begin{cases} 1 & \text{na pozici } (i,j) \text{ je typ } k \\ 0 & \text{na pozici } (i,j) \text{ není typ } k \end{cases},$$

$1 \leq i, j \leq n$, $1 \leq k \leq |K|$ a formule

(1) $\forall i, j : x_{ij1} \vee x_{ij2} \vee \ldots \vee x_{ij|K|}$, tedy na pozici (i,j) je aspoň jeden typ kachlíku

(2) $\forall i, j \; \forall k \neq k' : \neg x_{ijk} \vee \neg x_{ijk'}$, tedy na pozici (i,j) je nejvýšše jeden typ kachlíku

(3) $\forall 1 \leq i \leq n \; \forall 1 \leq j \leq n-1 : \neg x_{ijk} \vee \neg x_{i,j+1,k'}$ pro každou dvojici k, k' takovou, že k nemůže být vlevo od k'

(4) $\forall 1 \leq i \leq n-1 \; \forall 1 \leq j \leq n : \neg x_{ijk} \vee \neg x_{i+1,jk'}$ pro každou dvojici k, k' takovou, že k nemůže být nad k'

(5) $\forall j : \neg x_{1jk}$ pokud kachlík typu k nemůže být na pozici $(1,j)$

(6) $\forall j : \neg x_{njk}$ pokud kachlík typu k nemůže být na pozici (n,j)

(7) $\forall i : \neg x_{i1k}$ pokud kachlík typu k nemůže být na pozici $(i,1)$

(8) $\forall i : \neg x_{ink}$ pokud kachlík typu k nemůže být na pozici (i,n).

Formule $\mathcal{F}$ má délku $O(n^2|K|^2)$, tedy převod je polynomiální. ■

7.0.5 3-SAT

Definice. Kubická KNF $\mathcal{F}$ na n Booleovských proměnných. Existuje pravdivostní ohodnocení proměnných, které splňuje formuli $\mathcal{F}$?

Věta 7.5. *3-SAT je NP-úplný.*

Důkaz. 3-$SAT \in \mathbf{NP}$ a dokážeme, že $SAT \propto$ 3-SAT.

Nechť $\mathcal{F} = \wedge_{i=1}^{l}(\vee_{j=1}^{k_i} a_{ij}$ je zadání SAT. Zkonstruujeme kubickou KNF $\mathcal{F}'$ takovou, že $\mathcal{F}'$ je splnitelná právě tehdy, když $\mathcal{F}$ je splnitelná. Pro každou klauzuli tvaru $C_i = \vee_{j=1}^{k_i} a_{ij}$ z $\mathcal{F}$ pro kterou $k_i > 3$ zkonstruujeme KNF jako $\mathcal{F}_i = (a_{i1} \vee a_{i2} \vee y_{i1}) \wedge (\neg y_{i1} \vee a_{i3} \vee y_{i2}) \wedge \ldots \wedge (\neg y_{i,k_i-4} \vee a_{i,k_i-2} \vee y_{i,k_i-3}) \wedge (\neg y_{i,k_i-3} \vee a_{i,k_i-1} \vee a_{ik_i})$ a $\mathcal{F}'$ vznikne z $\mathcal{F}$ nahrazením každé C_i s $k_i > 3$ příslušnou podformulí.

Pokud je $\mathcal{F}$ splnitelná, pak v každé C_i existuje a_{ij} které je (příslušným pravdivostním ohodnocením) splněno, $\mathcal{F}_i$ splníme s ohodnocením nových proměnných $y_{it} = 1$ pro $1 \leq t \leq j-2$ a $y_{it} = 0$ pro $j-1 \leq t \leq k_i - 3$.

Pokud je $\mathcal{F}'$ splnitelná, pak restrikce příslušného splňujícího pravdivostního ohodnocení na "staré" proměnné splňuje $\mathcal{F}$ (neexistuje splňující ohodnocení $\mathcal{F}_i$ takové, že $\forall j \; a_{ij} = 0$).

Pokud $|C_i| = k$, pak $|\mathcal{F}_i| = k + 2(k-3) \leq 3k$ a tedy $|\mathcal{F}'| \leq 3|\mathcal{F}|$, tedy převod je polynomiální. ■

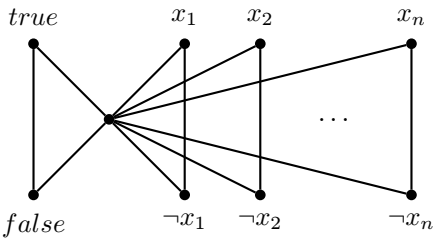
7.0.6 3-Barvení Grafu (3-BG)

Definice. Neorientovaný graf $G = (V, E)$. Lze obarvit vrcholy ve V třemi barvami tak, aby žádná hrana v E nebyla monochromatická?

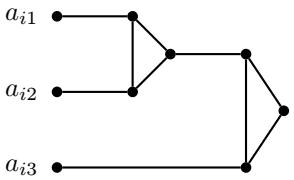
Věta 7.6. *3-BG je NP-úplný.*

Důkaz. 3-$BG \in$ **NP** a dokážeme, že 3-$SAT \propto$ 3-BG.

Nechť $\mathcal{F} = \wedge_{i=1}^{l}(a_{i1} \vee a_{i2} \vee a_{i3})$ na proměnných $x_1, x_2, \ldots, x_n$ je zadání 3-SAT (pokud by počet literálů v každé klauzuli byl menší než 3, mohu tuto doplnit tak aby obsahovala právě 3). Zkonstruujeme graf $G_{\mathcal{F}}$ jako



Ke každé $C_i = (a_{i1} \vee a_{i2} \vee a_{i3})$ přidáme do grafu následující podgraf



Graf $G_{\mathcal{F}}$ lze obarvit 3 barvami právě tehdy když formule $\mathcal{F}$ je splnitelná.

Pokud $|\mathcal{F}| = 3l$, pak $|G| = 2n+3+6l \leq 12l+3$ protože $n \leq 3l$, tedy převod je polynomiální. ■

7.0.7 Klika (KL)

Definice. Neorientovaný graf $G = (V, E)$ a přirozené číslo k. Existuje $V' \subseteq V$, $|V'| = k$, indukující úplný podgraf grafu G?

Věta 7.7. *KL je NP-úplný.*

Důkaz. $KL \in$ **NP** a dokážeme, že $SAT \propto KL$.

Nechť $\mathcal{F} = \wedge_{i=1}^{l}(\vee_{j=1}^{k_i} a_{ij}$ je zadání SAT. Zkonstruujeme $G_{\mathcal{F}} = (V_{\mathcal{F}}, E_{\mathcal{F}})$ tak, že $V_{\mathcal{F}} = \{a_{ij} | 1 \leq i \leq l, 1 \leq j \leq k_i\}$ a $E_{\mathcal{F}} = \{(a_{ij}, a_{i',j'} | (i \neq i') \& (a_{ij} \neq \neg a_{i'j'})\}$ a $k = l$. Platí, že $\mathcal{F}$ je splnitelná právě tehdy, když v $G_{\mathcal{F}}$ existuje klika velikosti l. V každé klauzuli vybereme jeden splněný literál, vrcholy odpovídající těmto literálům tvoří v $G_{\mathcal{F}}$ kliku velikosti l. Pokud v $G_{\mathcal{F}}$ existuje klika velikosti l, pak každý vrchol je z jiné klauzule a všechny literály odpovídající vrcholům z kliky lze splnit zároveň.

Platí $|\mathcal{F}| = |V_{\mathcal{F}}|$, tedy převod je polynomiální. ■

7.0.8 Nezávislá Množina (NM)

Definice. Neorientovaný graf $G = (V, E)$ a přirozené číslo q. Existuje $V' \subseteq V$, $|V'| = q$, taková, že uvnitř V' nejsou žádné hrany?

Věta 7.8. *NM je NP-úplný.*

Důkaz. $MN \in$ **NP** a dokážeme, že $KL \propto MN$.

MN je ekvivalentní KL na inverzním grafu. ■

7.0.9 Vrcholové Pokrytí (VP)

Definice. Neorientovaný graf $G = (V, E)$ a přirozené číslo r. Existuje $V' \subseteq V$, $|V'| = r$, taková, že každá hrana má ve V' alespoň jeden vrchol?

Věta 7.9. *VP je NP-úplný.*

Důkaz. $VP \in$ **NP** a dokážeme, že $NM \propto VP$.

Nechť $G = (V, E)$ a q je zadání NM. Stačí $G' = G$ zvolit a $r = |V| - q$ protože doplněk nezávislé množiny je vrcholového pokrytí a naopak doplněk vrcholového pokrytí je nezávislá množina. ■

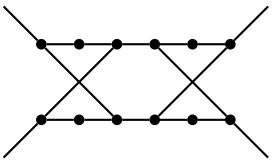
7.0.10 Hamiltonovská Kružnice (HK)

Definice. Neorientovaný graf $G = (V, E)$. Obsahuje G Hamiltonovskou kružnici, tj. jednoduchou kružnici, která prochází každým vrcholem právě jednou?

Věta 7.10. *HK je NP-úplný.*

Důkaz. $HK \in \mathbf{NP}$ a dokážeme, že $VP \propto HK$.

Nechť $G = (V, E)$ a r je zadáním VP, zkonstruujeme $G' = (V', E')$ zadání HK. Graf "plot"



Pokud do "plotu" vstoupí HK vlevo nahoře, tak vystoupí vpravo nahoře a jsou jen dva možné průchody – částečný a úplný.

Vytvořím $|V|$ "linek". Každá linka je potom posloupnost šestic vrcholů jednotlivých "plotů" (v libovolném pořadí). Každý z vrcholů $1, 2, \ldots, r$ spojím s prvním a posledním vrcholem na každé lince. Pokud $(i, j) \in E$ pak mezi linku i a linku j pověsím "plot" (ty se nesmějí překrývat). Existuje VP velikosti r, ta vybere r linek. Z vrcholu 1 jdu na 1. vybranou linku z jejíhož konce jdu na vrchol 2 a pak na 2. vybranou linku, atd., z vrcholu r jdu na r. vybranou linku a zpět do vrcholu 1. "Ploty" jejichž druhá strana leží na pokryté lince projdu částečně, ostatní úplně. Pokud existuje HK je postup obdobný.

Platí $|V'| = r + 12|E|$ kde $r \leq |V|$, tedy převod je polynomiální. ■

7.0.11 Obchodní Cestující (TSP)

Definice. Úplný neorientovaný graf $G = (V, E)$, váhy $w : E \to \mathbb{Z}_0^+$, a číslo $k \in \mathbb{Z}^+$. Existuje v G Hamiltonovská kružnice s celkovou váhou nejvýše k?

Věta 7.11. *TSP je NP-úplný.*

Důkaz. $TSP \in \mathbf{NP}$ a dokážeme, že $HK \propto TSP$.

Nechť $G = (V, E)$ je zadání HK. $G' = (V, V \times V)$, $w(e) = \begin{cases} 1 & e \in E \\ 2 & e \notin E \end{cases}$ a $k = |V|$.

Pro graf s n vrcholy a m hranami zkonstruujeme nový graf s n vrcholy a $n \cdot (n-1)/2$ hranami, tedy převod je polynomiální. ■

7.0.12 Součet Podmnožiny (SP)

Definice. Čísla $a_1, \ldots, a_n, b \in \mathbb{Z}^+$. Existuje množina indexů $S \subseteq \{1, \ldots, n\}$, taková, že $\sum_{i \in S} a_i = b$?

Věta 7.12. *SP je NP-úplný.*

Důkaz. $SP \in \mathbf{NP}$ a dokážeme, že $VP \propto SP$.

Nechť $G = (V, E)$ a číslo k je zadáním VP. Mějme incidenční matici D_G kde $(D_G)_{ij} = d_{ij}^G = \begin{cases} 1 & \text{pokud } v_i \text{ leží na } e_j \\ 0 & \text{jinak} \end{cases}$. Položme $x_i = 4^m + \sum_{j=0}^{m-1} d_{ij} 4^j$ pro $0 \leq i \leq n-1$, $y_j = 4^j$ pro $0 \leq j \leq m-1$ a $b = k \cdot 4^m + \sum_{j=0}^{m-1} 2 \cdot 4^j$.

Pro graf s n vrcholy a m hranami zkonstruuji čísla obsahující celkem $(n+m+1) \cdot (m+1) - 1$ "čtyřkových" cifer a "čtyřkový" zápis čísla k, tedy převod je polynomiální. ■

SP$(A, a_0, \ldots, a_n, b)$
Vstup: předpokládejme, že platí $a_1 \geq \ldots \geq a_n$, a že A je pole délky b
for $j \leftarrow 1$ to b **do**
$A(j) \leftarrow 0$
$a_0 \leftarrow b + 1$
end for
for $i \leftarrow 1$ to n **do**
$A(a_i) \leftarrow 1$
for $j \leftarrow b$ down to a_{i-1} **do**
if $A(j) = 1$ & $j + a_i \leq b$ **then**
$A(j + a_i) \leftarrow 1$
end if
end for
end for
return $A(b) = 1$

Tvrzení 7.13. *Po i-tém průchodu hlavním cyklem obsahuje pole A jedničky právě u těch indexů, které odpovídají součtům všech neprázdných podmnožin množiny $\{a_1, \ldots, a_i\}$, které jsou nejvýše rovny b.*

Důkaz. Dokážeme indukcí dle i. Pro $i = 1$ platí. Nechť platí pro $i - 1$. Vezměme libovolnou neprázdnou podmnožinu X množiny $\{a_1, \ldots, a_i\}$ takovou, že $\sum_{j \in X} a_j \leq b$. Pokud $i \notin X$, pak tvrzení platí dle indukčního předpokladu. Pokud $i \in X$, dle indukčního předpokladu po $i - 1$ průchodu tvrzení platí pro $X \setminus \{i\}$ a pokud $X \setminus \{i\} \neq \emptyset$, pak bude k $A(\sum_{j \in X \setminus \{i\}} a_j)$ přičteno a_i ve vnitřním cyklu. Pokud by $X \setminus \{i\} = \emptyset$, pak $A(a_i)$ bude přiřazeno zvlášť. ■

Pozorování 7.14. *Časová složitost algoritmu je $O(nb)$, což je exponenciální časová složitost vzhledem k binárně (ale také ternárně, dekadicky, ...) kódovanému vstupu, ale polynomiální časová složitost vzhledem k unárně kódovanému vstupu. Algoritmy s těmito vlastnostmi se nazývají* pseudopolynomiální.

7.1 Pseudopolynomiální algoritmy

Definice. Nechť je dán rozhodovací problém Q a jeho instance X. Definujme

$code(X)$ délka zápisu (počet bitů) instance X v binárním (či "vyšším") kódováním

$max(X)$ největší číslo v X (velikost čísla, ne délka jeho binárního zápisu).

Definice. Algoritmus řešící Q se nazývá *pseudopolynomiální*, pokud je jeho časová složitost při spuštění na vstupu X omezena polynomem v proměnných $code(X)$ a $max(X)$.

Poznámka 7.15. Každý polynomiální algoritmus je samozřejmě také pseudopolynomiální.

Pozorování 7.16. *Pokud je Q takový, že pro každou jeho instanci X platí $max(X) \leq p(code(X))$ pro nějaký polynom p, tak pro Q pojem polynomyálního a pseudopolynomiálního algoritmu splývá. Problémy, kde toto nenastává budeme nazývat číselné problémy.*

Definice. Rozhodovací problém Q se nazývá číselný, pokud neexistuje polynom p takový, že pro každou instanci X problému Q platí $max(X) \leq p(code(X))$.

Věta 7.17. *Nechť Q je NP-úplný problém, který není číselný. Potom pokud $\mathbf{P} \neq \mathbf{NP}$, tak Q nemůže být řešen pseudopolynomiálním algoritmem.*

Otázka. Je každý číslený problém řešitelný nějakým pseudopolynomiálním algoritmem? Ne (a typickým představitelům takových problémů se říká silně NP-těžké).

7.2 Silně NP-úplné problémy

Definice. Nechť je Q rozhodovací problém a p polynom. Symbolem Q_p označíme množinu instancí problému Q (tj. podproblém problému Q), pro které platí $max(X) \leq p(code(X))$, tj.

$$Q_p = \{X \in Q | max(X) \leq p(code(X))\}.$$

Věta 7.18. *Nechť je A pseudopolynomiální algoritmus řešící Q. Potom pro každý polynom p je A polynomiálním algoritmem řešícím Q_p.*

Definice. Rozhodovací problém Q se nazývá *silně NP-úplný*, pokud $Q \in \mathbf{NP}$ a existuje polynom p takový, že podproblém Q_p je NP-úplný.

Věta 7.19. *Nechť Q je silně NP-úplný problém. Potom pokud $\mathbf{P} \neq \mathbf{NP}$, tak Q nemůže být řešen pseudopolynomiálním algoritmem.*

Příklad (Obchodní cestující (TSP))**.** Je to číselný problém (váhy na hranách mohou být libovolně velké). Je silně NP-úplný neboť zůstává NP-úplný i když váhy omezíme (malou) konstantou.

Příklad (3-partition (3-P))**.** Buď $a_1, \ldots, a_{3m}, b \in \mathbb{N}$, taková že $\forall j : 1/4b < a_j < 1/2b$ a platí $\sum_{j=1}^{3m} a_j = mb$. Existuje $S_1, \ldots, S_m$ disjunktní rozdělení množiny $\{1, \ldots, 3m\}$ takové, že $\forall i : \sum_{j \in S_i} a_j = b$?
Toto je "čistě" číselný problém.

8 Početní úlohy

Definice. U rozhodovacích problémů se ptáme na existenci řešení, u početních úloh na počet řešení.

Značení. Γ je abeceda na kódování problému, Σ je abeceda na kódování certifikátu.

Definice. Funkce $w : \Sigma^* \to P(\Gamma^*)$, kde $P(\Gamma^*)$ je potenční množina množiny Γ^*, se nazývá *certifikační funkce* pro Σ a Γ. Každý prvek množiny $w(x)$ se nazývá *certifikát* pro instanci x. Rozhodovací problém A_w spojený s certifikační funkcí w je definován předpisem $A_w = \{x \in \Sigma^* | w(x) \neq \emptyset\}$ (tj. A_w je množina kódů těch instancí, pro které existuje alespoň jeden certifikát).

Příklad (Splnitelnost (SAT))**.** Σ je abeceda na kódování KNF, Γ je abeceda na kódování pravdivostních ohodnocení. Potom $w(x) = \{y | y$ splňuje $x\}$ a $A_w = \{x | x$ je splnitelná$\}$.

Příklad (Klika (KL))**.** Σ kóduje graf a číslo k, Γ kóduje množinu vrcholů "správné velikosti". Potom $w(x) = \{y | y$ je úplný podgraf zadání $x\}$ a $A_w = \{x |$v x existuje klika "správné velikosti"$\}$.

Definice. Třída #P je množina certifikačních funkcí w takových, že

(1) existuje deterministický algoritmus, který pro každé $x \in \Sigma^*$ a $y \in \Gamma^*$ ověří v polynomiálním čase (vzhledem k $|x| + |y|$) jestli $y \in w(x)$,

(2) existuje polynom p takový, že $\forall j \in w(x) : |y| \leq p(|x|)$ (pro různá w a p).

Věta 8.1.

(1) Pokud $w \in \#\mathbf{P}$, pak $A_w \in \mathbf{NP}$.

(2) Pokud $A \in \mathbf{NP}$, pak $\exists w \in \#\mathbf{P} : A = A_w$.

Důkaz.

(1) Chceme ukázat, že existuje NTS M rozpoznávající jazyk A_w v polynomiálním čase. M pracuje tak (na vstupu x), že uhádne y (polynomiálně velké vzhledem k x) (viz vlastnost (2)) a potom deterministicky ověří, zda $y \in w(x)$ (viz vlastnost (1)).

(2) Nechť M je NTS přijímající A v polynomiálním čase, tedy $\forall x \in A$: M přijme x v čase nejvýše $p(|x|)$. Buď $w(x) = \{y \in \Gamma^* | M$ přijme x po přijímající cestě s kódem y délky nejvýše $p(|x|)\}$. Vlastnost (2) plyne z definice. Zkonstruujeme DTS M' který zjistí zda $|y| \leq p(|x|)$, pokud ne, tak odmítne $(y \notin w(x))$. Pokud ano, M' simuluje M s tím, že používá y jako kód cesty ve stromě všech výpočtů stroje M. Tím je ověřena vlastnost (1).

■

Věta 8.2. *Pokud $w \in \#\mathbf{P}$ taková, že $\forall x \in \Sigma^*$ lze v polynomiálním čase spočítat $|w(x)|$, tak lze $\forall x \in \Sigma^*$ v polynomiálním čase rozhodnout zda $x \in A_w$ (tj. vyřešit příslušný problém ze třídy NP) i zda $x \in \overline{A}_w$ (tj. vyřešit příslušný doplňkový problém ze třídy co-NP).*

Důsledek 8.3. *Početní úlohy ($\#SAT, \#KLIKA, \#SP, \ldots$) odpovídající NPÚ problémům ($SAT, KLIKA, SP, \ldots$), jsou alespoň stejně tak těžké jako dané problémy.*

Otázka. Lze pro třídu #P zavést analogii polynomiálních transformací a úplnosti úloh?

Definice. Nechť $w : \Sigma^* \to P(\Gamma^*)$ a $v : \Pi^* \to P(\Delta^*)$ jsou certifikační funkce z #P. Polynomiální *redukce* z w na $v$je dvojice funkcí vyčíslitelných v polynomiálním čase $\sigma : \Sigma^* \to \Pi^*$ a $\tau : \mathbb{N} \to \mathbb{N}$ takových, že $\forall x \in \Sigma^* : |w(x)| = \tau(|v(\sigma(x))|)$.

Definice. Pokud $\forall n \in \mathbb{N} : \tau(n) = n$ (τ je identita), tak se příslušná funkce nazývá *šetrná*.

Definice. Certifikační funkce v je #P-úplná pokud

(1) $v \in \#\mathbf{P}$

(2) $\forall w \in \#\mathbf{P}$ existuje polynomiální redukce z w na v.

Věta 8.4. *$\#KACHL$ je #P-úplná úloha.*

Důkaz. Nechť $w \in \#\mathbf{P}$ libovolná a dokážeme, že existuje polynomiální redukce z w na $\#KACHL$.

Pokud $w \in \#\mathbf{P}$, pak existuje DTS M, který v polynomiální čase pro $\forall x \in \Sigma^*$ a $\forall y \in \Gamma^*$ ověří, zda $y \in w(x)$. Existuje NTS M' přijímající $x \in \Sigma^*$ právě tehdy, když $\exists y \in \Gamma^* : y \in w(x)$ (M' rozpoznává A_w) a takový, že $y \in w(x)$ odpovídají přijímajícím výpočtům M' délky nejvýšše $p(|x|)$. Z důkazu 7.1 víme, že pro každý NTS M' a vstupy x existuje instance t problému $KACHL$ taková, že M' přijímá x tehdy, a jen tehdy když t má legální vykachlíkování. Navíc platí, že přijímající výpočty M' délky nejvýšše $p(|x|)$ odpovídají legálnímu vykachlíkování instance t problému $KACHL$. Příslušná funkce $\sigma : \Sigma^* \to L(KACHL)$ taková, že A_w odpovídá $L(KACHL)_Y$, definuje šetrnou redukci z w na $\#KACHL$. ■

Věta 8.5. *$\#SAT$ je #P-úplná úloha.*

Důkaz. Vyplývá z transformace $KACHL \propto SAT$, která definuje šetrnou redukci. ■

Věta 8.6. *$\#3\text{-}SAT, \#KLIKA, \#SP$ jsou #P-úplné úlohy.*

Důkaz. Mírnými úpravami transformací použitých k důkazům NP-úplnosti lze získat šetrné redukce. ■

Pozorování 8.7. *Pro $w \in \#P$ a $x \in \Sigma^*$ spočítat $|w(x)|$ je alespoň tak těžké jako rozhodnout zda $x \in A_w$.*

Otázka. Existuje $w \in \#\mathbf{P}$, kde rozhodnout zda $x \in A_w$ je lehké (řešitelné v polynomiálním čase $\forall x \in \Sigma^*$), ale spočítat $|w(x)|$ je těžké (w je #P-úplná)?

Tvrzení 8.8. *Nechť $G = (U, V, E)$ je bipartitní graf kódovaný řetězcem x a nechť $y \in w(x)$ pokud y kóduje perfektní párování v G. Potom*

$$A_w = \{x | v\ x\ \textit{existuje perfektní párování}\}.$$

a platí

(1) Rozhodnout zda $x \in A_w$ lze v polynomiálním čase.

(2) Spočítat $|w(x)|$ je #P-úplná úloha.

Tvrzení 8.9. Permanent *čtvercové matice $A = (a_{ij})$ typu $n \times n$ je definován předpisem*

$$\mathrm{perm}(A) = \sum_{\sigma \in S_n} \Pi_{i=1}^{n} a_{i\sigma(i)},$$

kde S_n je cyklická grupa všech permutací řádu n.

Pozorování 8.10. *Permanent incidenční matice bipartitního grafu $G = (U, V, E)$ (kde $|U| = |V|$) je přesně roven počtu perfektních párování v grafu G.*

Důsledek 8.11. *Úloha spočítat permanent matice je #P-úplná i pro matice, které obsahují jako své prvky pouze čísla 0 a 1.*

Důkaz. Incidenční matice A velikosti $|U| \times |V|$ kde $(A)_{ij} = a_{ij} = \begin{cases} 1 & (i,j) \in E \\ 0 & (i,j) \notin E \end{cases}$. ■

9 Aproximační algoritmy

Aproximační algoritmy jsou typicky používány na řešení "velkých" instancí NP-těžkých optimalizačních problémů, pro které je nalezení optimálního řešení "beznadějné", tj. časově příliš náročné (pro "malé" instance lze nalézt optimální řešení "hrubou silou" v exponenciálním čase). Aproximační algoritmus má následující tři vlastnosti:

(1) Vrací (většinou) suboptimální řešení (někdy ale může vrátit i optimum).

(2) Dává odhad kvality vráceného řešení vzhledem k optimu.

(3) Běží v polynomiálním čase vzhledem k velikosti zadání.

Značení.

OPT optimální řešení

APR řešení vrácené aproximačním algoritmem

$f(Z)$ hodnota řešení Z (předpokládáme, že je vždy nezáporná)

Definice. Aproximační algoritmus A řeší optimalizační problém X s *poměrovou chybou* $r(n)$, pokud pro všechna zadání problému X velikosti n platí

$$\max\left\{\frac{f(APR)}{f(OPT)}, \frac{f(OPT)}{f(APR)}\right\} \leq r(n).$$

Tedy pro minimalizaci $f(OPT) \leq f(APR)$ a $\frac{f(APR)}{f(OPT)} \leq r(n)$, pro maximalizaci $f(OPT) \geq f(APR)$ a $\frac{f(OPT)}{f(APR)} \leq r(n)$.

Definice. Aproximační algoritmus A řeší optimalizační problém X s *relativní chybou* $e(n)$ pokud pro všechna zadání problému X velikosti n platí

$$\frac{|f(APR) - f(OPT)|}{f(OPT)} \leq e(n).$$

Tedy pro minimalizaci $\frac{f(APR)-f(OPT)}{f(OPT)} \leq e(n)$ což odpovídá $\frac{f(APT)}{f(OPT)} - 1 \leq e(n)$, pro maximalizaci $\frac{f(OPT)-f(APR)}{f(OPT)} \leq e(n)$ což odpovídá $1 - \frac{f(APR)}{f(OPT)}$.

Příklad (Optimalizační verze problému KL)**.** Příklad maximalizačního problému. Pro daný graf najít největší (měřeno počtem vrcholů) kliku v daném grafu. Aproximační algoritmus by musel poskutnout odhad (záruku kvality) tohoto typu $f(APR) \geq \frac{3}{4}(OPT)$, kde $f(X)$ je v tomto případě počet vrcholů (tj. velikost kliky) v řešení X.

Příklad (Optimalizační verze problému TSP)**.** Příklad minimalizačního problému. Pro daný úplný vážený neorientovaný graf najít nejkratší Hamiltonovskou kružnici (měřeno součtem délek hran) v daném grafu. Aproximační algoritmus by musel poskytovat odhad (záruku kvality) tohoto typu $f(APR) \leq 2f(OPT)$, kde $f(X)$ je v tomto případě délka Hamiltonovské kružnice v řešení X.

9.0.1 Úloha vrcholového pokrytí

Definice. Neorientovaný graf $G = (V, E)$. Úlohou je najít vrcholové pokrytí minimální velikosti, tj. najít $V' \subseteq V$ takové, že pro každé $(u, v) \in E$ platí $u \in V'$ nebo $v \in V'$ (nebo oboje), a navíc V' má minimální možnou kardinalitu.

Definice (Algoritmus A)**.** Opakovaně vyber v grafu vrchol nejvyššího stupně, přidej ho do postupně konstruovaného vrcholového pokrytí a odstraň ho z grafu spolu se všemi incidentními (a tedy pokrytými) hranami dokud nezbývá v grafu žádná hrana.

Tvrzení 9.1. *Algoritmus 9.0.1 nemá konstantní relativní (poměrovou) chybu.*

Důkaz. Graf Platí $f(APR) = \sum_{i=3}^{n} \lfloor \frac{n}{i} \rfloor + n \geq \sum_{i=3}^{n} \frac{n}{i} - (n-2) + n = n \sum_{i=1}^{n} \frac{1}{i} + 2 - n\frac{n}{2} \geq cn \ln n$ pro vhodné c, $f(OPT) = n$ a platí $\frac{f(APR)}{f(OPT)} \geq \frac{cn \ln n}{n} = c \ln n$. ■

Definice (Algoritmus B)**.** Opakovaně vyber v grafu libovolnou hranu (u, v) a dej jak u tak v do postupně konstruovaného vrcholového pokrytí a odstraň jak u, tak v z grafu spolu se všemi incidentními (a tedy pokrytými) hranami dokud nezbývá v grafu žádná hrana.

Tvrzení 9.2. *Algoritmus 9.0.1 má konstantní poměrovou chybu.*

Důkaz. Platí že množina vybraných hran E' je množina po dvou disjunktních hran. Tedy $f(APR) = 2|E'|$ a $f(OPT) \geq |E'|$ odkud platí $\frac{f(APR)}{f(OPT)} \leq 2$. ■

9.0.2 Úloha obchodního cestujícího

Definice. Úplný vážený neorientovaný graf $G = (V, E)$ a váhová funkce $c : E \to \mathbb{Z}^+ \cup \{0\}$. Úlohou je najít v G Hamiltonovskou kružnici nejmenší celkové váhy (délky).

Definice (Obchodní cestující s trojúhelníkovou nerovností)**.** Úlohou je najít v G Hamiltonovskou kružnici nejmenší celkové váhy a zároveň musí platit $\forall u, v, w \in V : c(u, w) \leq c(u, v) + c(v, w)$. Tento problém je NP-těžký.

Definice (Algoritmus C)**.**

(1) Najdi minimální kostru grafu G.

(2) Vyber libovolný vrchol grafu G a spusť z něj na nalezené kostře algoritmus DFS, který očísluje vrcholy v preorder pořadí.

(3) Výsledná Hamiltonovská kružnice je dána pořadím (permutací) z kroku (2).

Poznámka 9.3. Pokud je v kroku (1) použit Primův (Jarníkův) algoritmus, tak celý algoritmus běží v čase $O(|E|) = O(|V|^2)$.

Věta 9.4. *Algoritmus 9.0.2 má konstatní poměrovou chybu $r(n) \leq 2$.*

Důkaz. Buď T minimální kostra, potom $f(OPT) \geq f(T)$. W buď úplná procházka po kostře T, potom $f(W) = 2f(T)$. Zároveň platí $f(W) \geq f(APR)$ díky trojúhelníkové nerovnosti. Odtud vyplývá $\frac{f(APR)}{f(OPT)} \leq \frac{f(W)}{f(OPT)} \leq \frac{f(W)}{f(T)} = \frac{2f(T)}{f(T)} = 2$. ■

Věta 9.5. *Nechť $R \geq 1$ je libovolná konstanta. Potom pokud $\mathbf{P} \neq \mathbf{NP}$, tak neexistuje polynomiální aproximační algoritmus řešící obecný případ obchodního cestujícího s poměrovou chybou nejvýše R.*

Důkaz. Nechť A je polynomiální aproximační algoritmus s poměrovou chybou R řešící obecný případ problému TSP. Ukážeme, že A lze použít k vyřešení problému HK v polymoniálním čase odkud vyplývá $\mathbf{P} = \mathbf{NP}$.

Nechť $G = (V, E)$ je zadáním HK. Definujme graf $K_n = (V, E' = V \times V)$ a $w : E' \to \mathbb{N}$ (zadání TSP) takto $w(e) = \begin{cases} 1 & e \in E \\ nR & e \in E' \setminus E \end{cases}$. Nyní je vidět, že mohou nastat dva případy

(1) G obsahuje hamiltonovskou kružnici; v takovém případě optimální hodnota pro TSP je n, potom ale A musí vrátit hamiltonovskou kružnici s celkovou váhou menší nebo rovnu nR, odkud vyplývá, že A musí vrátit hamiltonovskou kružnici s váhou právě n, protože každá kružnice která má váhu různou od n má váhu větší než nR a tedy optimální hodnota pro TSP je větší než nR,

(2) G neobsahuje hamiltonovskou kružnici, tedy optimální hodnota pro TSP je větší než nR a tedy A vrátí hodnotu větší než A.

■

Důsledek 9.6 (O existenci neaproximovatelných úloh)**.** *Existují NP-těžké optimalizační úlohy, pro které neexistují polynomiální aproximační algoritmy s konstantní poměrovou chybou (pokud $\mathbf{P} \neq \mathbf{NP}$).*

Pozorování 9.7. *Existují NP-těžké optimalizační úkoly, které lze aproximovat s libovolně malou relativní chybou (poměrovou chybou libovolně blízko 1) s tím, že čím menší je požadovaná chyba tím vyšší je časová složitost aproximačního algoritmu.*

10 Aproximační schémata

Definice. *Aproximační schéma* (AS) pro optimalizační úlohu X je algoritmus, jehož vstupem je zadání Y úlohy X a (racionální) číslo $e > 0$, který pro libovolné pevné e pracuje jako aproximační algoritmus pro úlohu X s relativní chybou e.

Poznámka 10.1. Doba běhu může být exponenciální jak ve velikosti zadání Y tak v $1/e$.

Definice. *Polynomiální aproximační schéma* (PAS) pro optimalizační úlohu X je AS, jehož časová složitost je polynomiální vzhledem k velikosti zadání Y úlohy X.

Poznámka 10.2. Doba běhu může být stále ještě exponenciální vzhledem k $1/e$.

Definice. *Úplně polynomiální aproximační schéma* (ÚPAS) pro optimalizační úlohu X je PAS, jehož časová složitost je polynomiální také vzhledem k k $1/e$.

10.1 Úloha součtu podmnožiny (optimalizační verze)

Definice. Vstupem je množina přirozených čísel $A = \{x_1, \ldots, x_n\}$ a přirozené číslo t. Úlohou pak najít množinu indexů $S \subseteq \{1, \ldots, n\}$ takovou, že $sum = \sum_{i \in S} x_i$ je co největší při platnosti podmínky $sum \leq t$.

10.1.1 Pseudopolynomiální algoritmus pro SP

Značení. Nechť L je uspořádaný seznam přirozených čísel $a_1, \ldots, a_n$. Pak $L+x$, kde x je přirozené číslo je uspořádaný seznam přirozených čísel $a_1 + x, \ldots, a_n + x$.

SUM(A, t)
$L_0 \leftarrow (0)$ {seznam délky 1 obsahující číslo 0}
for $i \leftarrow 1$ to n **do**
  $L_i \leftarrow$ **MERGE**$(L_i - 1, L_i - 1 + x_i)$ {**MERGE** slije oba seznamy, čísla větší než t zahodí}
end for
return největší prvek v L_n

Věta 10.3. *Seznam L_i pro $1 \leq i \leq n$ je uspořádaný seznam obsahující součty všech podmnožin množiny $A = \{x_1, \ldots, x_n\}$, které jsou menší nebo rovny číslu t.*

Důkaz. Indukcí podle i. ■

Pozorování 10.4. *Složitost je v každém případě $O(|L_1| + \ldots + |L_n|)$. Pokud jsou v seznamech drženy duplicitní hodnoty tak (v nejhorším případě) $\Omega(2^n)$, ale pokud jsou duplicity v* **MERGE** *vyházeny, tak $O(nt)$. Algoritmus je polynomiální pokud $t \leq p(n)$ nebo $\forall i : x_i \leq p(n)$ pro nějaký polynom p.*

10.1.2 Prořezávání seznamů

Definice. Nechť je dáno $0 < d < 1$. Prořezat seznam L parametrem d znamená odebrat z L co nejvíce prvků tak, že pro každý odstraněný prvek y existuje v prořezaném seznamu L' prvek z takový, že $(1 - d)y \leq z \leq y$.

CUT(L, d)
$L' \leftarrow (y_1)$
$last \leftarrow y_1$
for $i \leftarrow 2$ to $|L|$ **do**
  if $last < (1 - d)y_i$ **then**
    $L' \leftarrow L' \cup \{y_i\}$
    $last \leftarrow y_i$
  end if
end for
return L'

Pozorování 10.5. *Časová složitost je $\Omega|L|$.*

10.1.3 ÚPAS pro SP

Definice. Vstupem je množina přirozených čísel $A = \{x_1, \ldots, x_n\}$, přirozené číslo t a aproximační parametr e.

Pozorování 10.6. *Časová složitost je $\Omega(|L_1| + \ldots + |L_n|)$.*

APPROX-SP(A, t, e)
$L_0 \leftarrow (0)$ {seznam délky 1 obsahující číslo 0}
for $i \leftarrow 1$ to n **do**
  $L_i \leftarrow$ **MERGE**$(L_i - 1, L_i - 1 + x_i)$ {**MERGE** slije oba seznamy, čísla větší než t zahodí}
  $L_i \leftarrow$ **CUT**$(L_i, e/n)$
end for
return největší prvek v L_n

Pozorování 10.7. *Opakovaným prořezáváním se chyba může postupně zvětšovat, ale e/n je dostatečně malý "prořezávací parametr", aby celková relativní chyba "nasčítaná" přes n iterací byla nejvýše e.*

Věta 10.8. *Algoritmus* **APPROX-SP** *je ÚPAS pro optimalizační úlohu SP.*

Značení. y^* je optimální hodnota, z je hodnota vrácená algoritmem **APPROX-SP**.

Poznámka 10.9. Chceme ukázat, že $(1–e)y^* \leq z \leq y^*$.

Lemma 10.10. *Nechť $y \leq t$ je součet nějaké podmnožiny množiny $\{x_1, \ldots, x_i\}$. Pak na konci i-té iterace algoritmu* **APPROX-SP** *existuje $w \in L_i$ (tj. w je v prořezaném seznamu L_i) takové, že platí $(1–e/n)^i y \leq w \leq y$.*

Důkaz. Indukcí podle i. Pro $i = 1$ v prořezaném L_1 platí. Nechť tvrzení platí pro $i - 1$ a nechť $y \leq t$ je součet nějaké podmnožiny čísel $\{x_1, \ldots, x_i\}$. Nyní mohou nastat dvě možnosti.

(1) Nechť x_i není ve vybrané množině. Z indukčního předpokladu $\exists w : (1 - e/n)^{i-1} y \leq w \leq y$ takové, že w je v prořezaném L_{i-1}
   - w přežije prořezávání L_i, tedy $(1 - e/n)^i y \leq (1 - e/n)^{i-1} y \leq w \leq y$
   - w nepřežije prořezávání L_i, tedy $\exists w' \in L_i : (1 - e/n)w \leq w' \leq w$ (po prořezání) odkud $(1 - e/n)^i y \leq (1 - e/n)w \leq w' \leq w \leq y$.

(2) Nechť x_i je ve vybrané množině. Z indukčního předpokladu víme, že $\exists w \in L_{i-1} : (1 - e/n)^{i-1}(y - x_i) \leq w \leq y - x_i$ (po prořezání), po **MERGE**$(L_{i-1}, L_{i-1} + x_i)$ je v L_i prvek $w + x_i \leq y \leq y$
   - $w + x_i$ přežije prořezávání L_i, tedy $(1 - e/n)^{i-1}(y - x_i) + x_i \leq w + x_i \leq y - x_i + x_i = y$ což je rovno $(1 - e/n)^{i-1} y + [1 - (1 - e/n)^{i-1}]x_i \leq w + x_i \leq y$ odkud $(1 - e/n)^i y \leq (1 - e/n)^{i-1} y \leq w + x_i \leq y$
   - $w + x_i$ nepřežije prořezávání L_i, tedy $\exists w' \in L_i : (1 - e/n)(w + x_i) \leq w' \leq w + x_i$ (po prořezání) odkud $(1 - e/n)(1 - e/n)^{i-1} y \leq (1 - e/n)(w + x_i) \leq w' \leq w + x_i \leq y$.

■

Důsledek 10.11. *Existuje $w \in L_n$ takové, že $(1–e/n)^n y^* \leq w \leq y^*$ a číslo z vrácené algoritmem* **APPROX-SP** *je největší takové w.*

Lemma 10.12. *$\forall n > 1$ platí $(1–e) < (1–e/n)^n$ a tudíž $(1–e)y^* \leq z \leq y^*$.*

Důkaz. Stačí dokázat, že $f(n) = (1 - e/n)^n$ je rostoucí na intervalu $[1, \infty)$ (protože $f(1) = (1-e)$). Platí $f(n) = e^{n \ln(1-e/n)}$ a $\frac{\partial f}{\partial n} = (1-e/n)^n \left[\ln(1 - e/n) + n \frac{1}{1-e/n} \frac{e}{n^2}\right]$. Chceme dokázat, že $\ln(1 - e/n) + n \frac{1}{1-e/n} \frac{e}{n^2} > 0$. Použijeme substituci $z = \frac{1}{1-e/n}$ a $\frac{e}{n} = 1 - \frac{1}{z}$, tedy $\ln \frac{1}{z} + z(1 - \frac{1}{z}) > 0$ odkud $z - 1 > \ln z$. Pokud $n \in [1, \infty)$, pak $z \in (1, \frac{1}{1-e}]$ a tedy $z > 1 + \ln z$ platí což bylo dokázat. ■

Poznámka 10.13. Víme, že časová složitost **APPROX-SP** je $\Theta(|L_1| + \ldots + |L_n|)$ a chceme ukázat, že je také $O(p(n, \log t, 1/e))$ pro nějaký polynom p ve třech proměnných.

Lemma 10.14. *Platí* $\forall i : |L_i| \leq (n \ln t)/e$.

Důkaz. V L_i po prořezání platí, že pokud $v < w$ jsou po sobě jdoucí prvky, tak musí platit $(1 - e/n)w > v$, tedy $\frac{w}{v} > \frac{1}{1-e/n}$. "Nejhustší" možné L_i je geometrická posloupnost s kvocientem $\frac{1}{1-e/n}$ odkud plyne $|L_i| \leq \log_{\frac{1}{1-e/n}} t$. Nyní stačí dokázat, že $\log_{\frac{1}{1-e/n}} t = \frac{\ln t}{\ln \frac{1}{1-e/n}} \leq \frac{n \ln t}{e}$, tedy $\frac{1}{\ln \frac{1}{1-e/n}} \leq \frac{n}{e}$ odkud $-e/n \geq \ln 1 - e/n$. Pro $x > -1$ platí $x \geq \ln 1 + x$ a vezmeme $x = -e/n$. ■

Tvrzení 10.15. *Algoritmus* **APPROX-SP** *má časovou složitost* $O((n^2 \log t)/e)$.