Složitost I - cvičení

Martin Všetička

1 Úvod

Pár důležitých pojmů, které je dobré si ujasnit před čtením příkladů.

1.1 NP a co-NP

NP: decision problems where at least the YES-instances have short proofs (that can be checked in polynomial-time) that the answer is YES. Q is in NP if there is a verifier $V(I, X)$ such that:

- If I is a YES-instance, then there exists X such that $V(I, X) =$ YES.
- If I is a NO-instance, then for all X, $V(I, X) = NO$.

and furthermore the length of X and the running time of V are poly in $|I|$.

co-NP: Problems where the the NO-instances have short proofs. (E.g.,given two circuits, C_1, C_2, do they compute the same function?). Formally, Q is in co-NP if there is a verifier $V(I, X)$ such that:

- If I is a YES-instance, then for all X, $V(I, X) =$ YES.
- If I is a NO-instance, then there exists X such that $V(I, X) =$ NO.

and furthermore the length of X and the running time of V are poly in $|I|$.

NP-completeness: Problem Q is *NP-complete* if:

1. Q is in NP, and
2. $Q' \leq_p Q$ for any other Q' in NP.

If Q just satisfies (2) then it's called *NP-hard*.

1.2 Redukce problému A na problém B

To reduce problem A to problem B we want a function f that takes instances of A to instances of B such that:

1. if x is a yes-instance of A then $f(x)$ is a yes-instance of B
2. if x is a no-instance of A then $f(x)$ is a no-instance of B
3. f can be computed in polynomial time.

So, if we had an algorithm for B, we could using it to solve A by running it on $f(x)$.

1. cvičení

Def $S = \{1, \ldots, n\}$ značí n úloh, pro které budeme hledat rozvrh.

Def Úlohy i a j jsou *kompatibilní*, pokud $[s_i, f_i) \cap [s_j, f_j) = \emptyset$. s_i (resp. f_i) značí startovní (resp. koncový) čas úlohy i.

Poznámka Hledání kompatibilních úloh je důležité například pro plánování úloh na jednoprocesorovém počítači.

Příklad Je dána množina úkolů $S = \{1, 2, \ldots, n\}$ a pro každý úkol i čas jeho zahájení $s(i)$ a jeho ukončení $f(i)$. Úkoly i a j jsou kompatibilní pokud mají polouzavřené intervaly $[s(i), f(i))$ a $[s(j), f(j))$ prázdný průnik. Najděte co největší množinu (po dvou) kompatibilních úkolů.

1. Na přednášce byl předveden hladový algoritmus řešící tento problém. Úkolem zde je dokázat, že ne každý hladový algoritmus povede k nalezení optima. Zkonstruujte protipříklady na následující hladové volby:
   (a) stejné jako na přednášce, ale úkoly setřídit podle $s(i)$
   (b) algoritmus v každém kroku vybere nejkratší úkol z těch, které jsou kompatibilní se všemi již vybranými

(c) algoritmus v každém kroku vybere takový úkol, který je kompatibilní se všemi již vybranými, a který se překrývá s nejmenším počtem úkolů z těch, které jsou kompatibilní se všemi již vybranými

2. Problém má stejná vstupní data jako dříve, ale tentokrát chceme rozvrhnout všechny úkoly za použití co nejmenšího počtu „zdrojů“ (např. posluchráren pokud úkoly = přednášky), tj. chceme rozdělit úkoly do co nejmenšího počtu kompatibilních podmnožin. Rozmyslete, zda lze pro řešení tohoto problému použít „iteraci“ algoritmu z přednášky (tj. nalezneme největší kompatibilní podmnožinu, ve zbytku opět nalezneme největší kompatibilní podmnožinu, atd. dokud zbývají nezařazené úkoly). Zkuste zkonstruovat jiný hladový algoritmus pro tento problém a dokažte jeho korektnost.

Obecný hladový algoritmus:

Dokud $S \neq \emptyset$ opakuj:

1. Podle KRITÉRIA vyber $x \in S$ a přidej x do řešení.
2. Z S odeber úlohy nekompatibilní s x.

Řešení:

1. ad 1.

   (a) Vybíráme vždy úkol s nejmenší hodnotu s_i.
   Protipříklad:

   ```
   [----- 1 ------]
    [- 2 -][- 3 -]
   ```

   Vybereme úkol 1, přestože bychom měli vybrat úkoly 2 a 3 (maximalizujeme kardinalitu množiny kompatibilních úkolů, ne trvání úloh!).

   (b) Nejkratší $f_i - s_i$.
   Protipříklad:

```
     [- 3 -]
[-- 1 --][-- 2 --]
```

Vybereme úkol 3 (čímž zabijeme úkoly 1 a 2), přestože bychom měli vybrat úkoly 1 a 2.

(c) Úkol, který se překrývá s nejmenším počtem úkolů z S.
Protipříklad:

```
  -----           -----
4 -----           -----  4
  -----   4    4  -----
  -- ------- ------- --  <----
         -------
            2
```

Nejlepší výsledek jsou čtyři úlohy naznačené šipkou. Když však vybereme úlohu, která se překrývá s dvěma jinými úlohami, tak zabijeme dvě úlohy a již můžeme získat pouze nejlepší rozvrh se třemi úlohami.

(d) Největší s_i (resp. nejmenší f_i). Toto již funguje! Je snadné si rozmyslet, že pokud máme optimální rozvrh O, který nepoužívá úlohu i s minimálním časem f_i, pak můžeme z O odebrat 1. úlohu a přidat do O úlohu i a stále máme optimální rozvrh.

2. ad 2.

   Jak by nás napadlo, že to jde dělat:

   - Iterativní použití kritéria (d) z minulého příkladu.
     Protipříklad:

```
  [--- 1 ---] [--- 2 ---]
                  [--- 3 ---]
[------ 4 ------]
```

Evidentně jdou dát úkoly do dvou kompatibilních množin $S_1 = \{1,2\}$ a $S_2 = \{4,3\}$. Náš algoritmus však vezme[1] úkol 3, pak úkol 1, čímž vznikne $S_1 = \{3,1\}$ a pak vytvoří $S_2 = \{2\}$ a následně $S_3 = \{4\}$. Algoritmus tedy vytvořil 3 množiny ačkoli stačí 2.

- Iterativní použití blackboxu, který dokáže vyřešit remízy v předchozím algoritmu, vybírajícího maximální kompatibilní množiny.

  Protipříklad:

```
     [-- 1 --] [-- 2 --] [--------- 3 -----------]
 [---------- 4 ------------]   [-- 5 --] [-- 6 --]
```

- Iterativní použití kritéria (c) s rozdílem, že koliduje s největším počtem úloh

  Protipříklad:

```
        3           2     1   1
[-------------][------][--][--]
[--][--][-------][------------]
 1   1      2           3
```

  Čísla udávají s kolika úlohami daná úloha koliduje. Je vidět, že opět algoritmus vrátí tři množiny úloh, kdežto evidetně je možné je rozdělit do dvou.

- Uspořádáme úlohy podle rostoucích s_i a zpracováváme je v tomto pořadí. Při zpracování úkolu se startovním časem s_i provedeme to, že jej přiřadíme do libovolné z již vytvořených kompatibilních skupin, ke které ho přiřadit lze a pokud taková skupina neexistuje, tak úkol "založí" novou skupinu.

  Formálně: Přidáváme úkol i. Pokud $\forall j \in \{1,\dots,k-1\}$ existuje úkol l v j-té skupině takový, že $s_l \leq s_i \leq f_l$. Tudíž nemůže existovat méně než k skupin úkolů a musíme založit k-tou skupinu úkolů.

[1] Vybíráme dle největšího s_i.

Příklad:

```
1 [-----]    [-----]
2      [-------]
3
4
.
.
.
k                 [-----]
```

Tento algoritmus tedy FUNGUJE.

Odbočka: Představme si, že úkoly odpovídají vrcholům grafu. Hrany odpovídají nekompatibilitám dvojic úkolů. Pak náš problém můžeme převést na problém barvení grafu. Najít chromatické číslo grafu je NP-úplný problém. My jsme však schopni náš problém s rozvrhy řešit v polynomiálním čase. To je divné, ne? Není, protože ne každý graf lze převést na rozvrh, například kružnice C_4 nejde. Rozvrhy odpovídají pouze jedné skupině grafů, kterým se říká *intervalové* a u nich lze najít chromatické číslo rychle.

Příklad V následujících příkladech dokažte, že daná struktura je matroid:

1. Nechť S je konečná neprázdná množina o n prvcích a k je přirozené číslo menší než n. Nechť I je množina všech podmnožin množiny S o velikosti nejvýše k. Dokažte, že (S, I) je matroid.

2. Nechť S je konečná neprázdná množina a nechť $S_1, S_2, \dots, S_n$ je rozdělení množiny S na po dvou disjunktní podmnožiny. Nechť $I = \{A | \forall i |A \cap S_i| \leq 1\}$. Dokažte, že (S, I) je matroid.

3. Nechť (S, I) je matroid a nechť

$$J = \{A \,|\, S \setminus A \text{ obsahuje nějakou maximální množinu z } I\}$$

Dokažte, že (S, J) je matroid.

Řešení

1. Nechť S je konečná a neprázdná. Nechť $I_k = \{A \subseteq S \mid |A| \leq k\}$.

   (a) S konečná, neprázdná – ANO

   (b) Dědičná vlastnost je zřejmá přímo z definice I_k. Jestliže mám $B \subseteq S$, pak $A \subseteq B$ má velikost určitě $\leq k$.

   (c) Ověřujeme: Jestliže $A \in I_k$, $B \in I_k$ a $|A| < |B| \Rightarrow \exists x \in B \setminus A$ tak, že $A \cup \{x\} \in I_k$. Takové x existuje, lze vzít libovolné z $B \setminus A$.

   Všechny tři vlastnosti matroidu jsme ověřili a tedy (S, I_k) je matroid.

2. - Nechť S je konečná a neprázdná.
   - Nechť $S_1, \ldots, S_n$ je rozklad S takový, že $\forall i \neq j$ platí $S_i \cap S_j = \emptyset$ a $\bigcup_{i=1}^{n} S_i = S$.
   - Nechť $I = \{A \subseteq S \mid \forall i : |A \cap S_i| \leq 1\}$.

   (a) S konečná, neprázdná – ANO

   (b) Dědičná vlastnost platí. Jestliže mám $B \in I$, pak pro $A \subseteq B$ určitě platí $\forall i : |A \cap S_i| \leq 1$ – ANO.

   (c) Výměnná vlastnost: Máme $A \in I$ a $B \in I$ takové, že $|A| < |B|$. Chceme ukázat, že existuje alespoň jedno S_i takové, že se s ním protíná B a ne A.

$$
\begin{aligned}
|B| > |A| &\Rightarrow |\{i \mid |B \cap S_i| = 1\}| > |\{i \mid |A \cap S_i| = 1\}| \\
&\Rightarrow \exists i : (|B \cap S_i| = 1) \& (|A \cap S_i| = 0) \\
&\Rightarrow \text{Nechť } x \in B \cap S_i \text{ pak } A \cup \{x\} \in I
\end{aligned}
$$

3. Nechť $M = (S, I)$ je matroid. Chceme ověřit, že $M' = (S, I')$ je také matroid, kde":

   $I' = \{A \subseteq S \mid S \backslash A$ obsahuje nějakou maximální nezávislou množinu z $I\}$.

   Ověřujeme vlastnosti matroidu:

   (a) S je neprázdná, protože M je matroid.

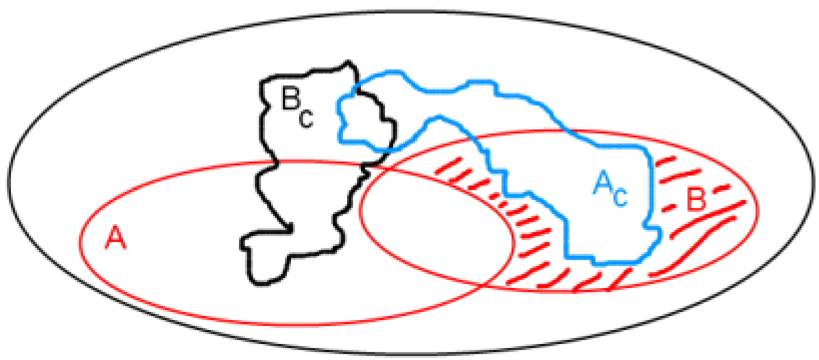
(b) Dědičná vlastnost: triviální, pro množinu $A \in I'$ existuje maximální nezávislá množina A_C v $S \setminus A$ a pro podmnožiny A si mohu vzít tu samou A_C.

[A_C vyjadřuje komplement k množině A, ale nejde o množinový komplement, pouze to označuje maximální nezávislou množinu, která existuje pro A.]

(c) Výměnná vlastnost:

Je nutno ukázat: $A, B \in I'$, $|A| < |B| \Rightarrow \exists x \in B \setminus A$ tak, že $A \cup \{x\} \in I'$

i. Jestliže existuje prvek $x \in B \setminus A$ tak že $x \notin A_C$, pak x je hledaným prvkem.



Kandidáti na prvek x jsou vyšrafováni červeně.

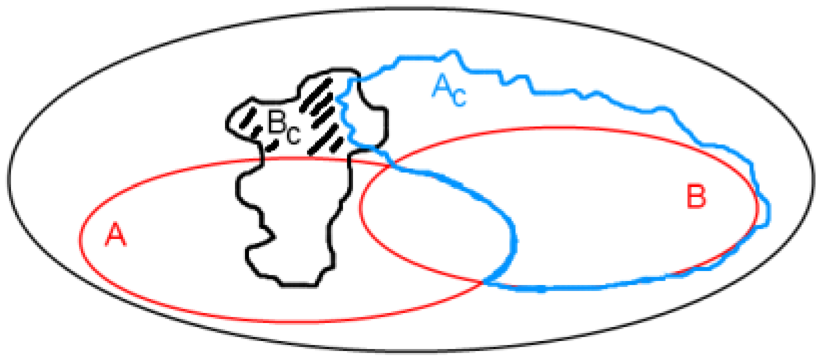
ii. Neexistuje prvek x z případu i., tedy platí $B \setminus A \subseteq A_C$.

Pozorování Jestliže $B \setminus A \subseteq A_C$, pak existuje prvek $y \in B_C$ takový, že $y \notin A_C$ a zároveň $y \notin A$.

Důkaz.

- Z předpokladu $|A| < |B|$ plyne $|A \setminus (A \cap B)| < |B \setminus (A \cap B)|$. Z předchozí nerovnosti a předpokladu $B \setminus A \subseteq A_C$ dále vyplývá, že existuje prvek z z množiny B_C takový, že $z \notin A$.
- Pokud by každý prvek množiny B_C, který neleží v A, patřil do A_C, pak by $|A_C| \neq |B_C|$ – SPOR.

□



- Nechť $B_{CC} = B_C \setminus A$
- Nechť $A_{CC} = A_C \setminus B$
- Množiny B_{CC} a A_{CC} jsou nezávislé množiny matroidu M dle dědičné vlastnosti.
- Víme, že $|B_{CC}| > |A_{CC}|$.
- Využijeme výměnnou vlastnost matroidu M, dostaneme $A_{CC2} = A_C \cup \{t\}$, kde $t \in B_{CC}$.
- Nyní chci doplnit A_{CC2} na velikost A_C, provádím tedy výměny mezit A_{CC2} a A_C, dokud nemá A_{CC2} stejnou kardinalitu jako A_C.
- Hledaným prvkem x je prvek z množiny $A_C \setminus A_{CC2}$, o které jsme dokázali, že je neprázdná.

2. cvičení

Příklad 1 Nechť je orientovaný graf $G = (V, E)$, kde $|V| = n$, zadán maticí sousednosti. Navrhněte algoritmus, který zjistí zda G obsahuje stok, tj. vrchol x takový, že

- pro vstupní stupeň x platí: $indegree(x) = n - 1$ a
- pro výstupní stupeň x platí $outdegree(x) = 0$,

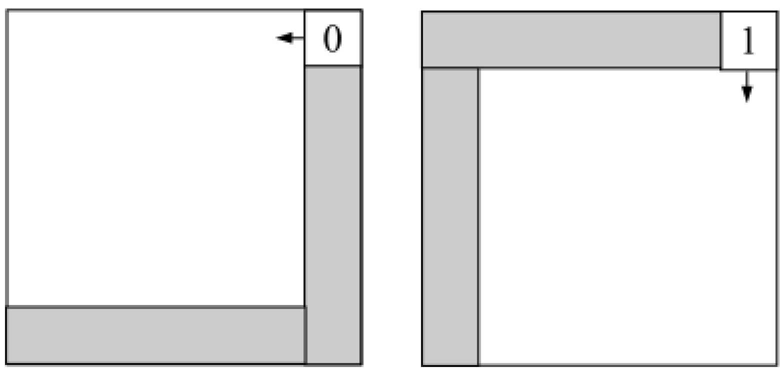
přičemž algoritmus smí použít (přečíst) pouze O(n) prvků matice. Předpokládejme, že před zahájením algoritmu je již celá matice načtena do paměti.

Def: Matice sousednosti A pro graf $G = (V, E)$ je matice $|V| \times |V|$ taková, že platí:

$$(i, j) \in E \Rightarrow A_{i,j} = 1$$
$$(i, j) \notin E \Rightarrow A_{i,j} = 0$$

Poznámka: Máme n^2 dat a chceme algoritmus s časovou složitostí $O(n)$. Takže se nemůžeme ani podívat na všechny prvky.

Řešení Úlohu jde vyřešit pomocí vybraného průchodu prvků matice.



(a) $A_{1,n} = 0$ (b) $A_{1,n} = 1$

Obrázek 1: Dvě možné výchozí pozice

Začneme v pravém horním rohu[2] matice sousednosti. První (resp. druhá) pozice na obrázku 1 vyznačují, na který další prvek matice se podíváme, jestliže na výchozí pozici je 0 (resp. 1). Proč zrovna takto?

- Pokud je na pozici $A_{i,j} = 0$, pak vrchol j nemůže být stok, protože do něj nevede hrana (i, j) a bude tedy platit $indegree(j) \leq n - 2$.
- Pokud je na pozici $A_{i,j} = 1$, pak vrchol i nemůže být stok, protože z něj vede hrana (i, j) a určitě bude mít $outdegree(i) \geq 1$.

V každém kroku procházky tedy dokážeme vyloučit jeden vrchol, který nemůže být stokem. Stačí tedy projít $n - 1$ pozic v matici A a o posledním vrcholu rozhodnout, zda je nebo není stokem, jelikož je to jediný kandidát.

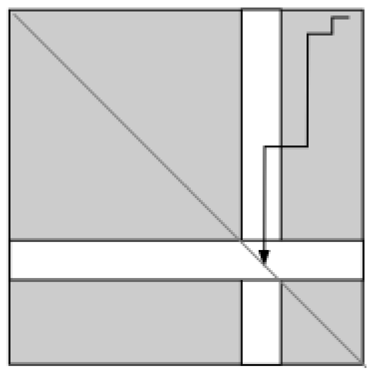
[2] Tedy na pozici $[0, n - 1]$, indexujeme od 0 do $n - 1$.

Po $n-1$ krocích skončíme na diagonále - proč? Uděláme z pravého horního rohu matice D kroků dolů a L doleva[3]. Budeme na pozici $[D, (n-1) - L]$. Protože $D = (n-1) - L$, tak po $n-1$ krocích skončíme na prvku

$$[(n-1) - L, (n-1) - L],$$

což je prvek diagonály.

Prvek, na kterém skončí procházka maticí, vidíme na následujícím obrázku.



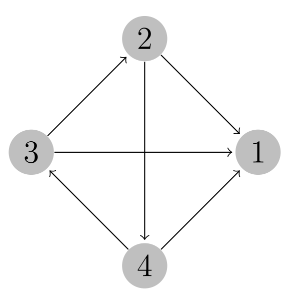
Obrázek 2: Konec procházky maticí

Tento prvek je jediný kandidát na stok a proto ho ověříme (tj. zda bílý řádek tvoří samé nuly a bílý sloupec samé jedničky (kromě prvku na diagonále)).

Složitost aloritmu:

$$T(n) = \Theta((n-1) + (n-2) + (n-1)) = \Theta(n)$$

Příklad: Mějme graf:



[3] Platí, že $D + L = n - 1$.

jeho matice sousednosti vypadá takto:

	1	2	3	4
1	0	0	0	0
2	1	0	0	1
3	1	1	0	0
4	1	0	1	0

Vrchol 1 je stok.

Příklad 2 Orientovaný graf G se nazývá *polosouvislý*, pokud pro každé dva vrcholy x, y existuje v G orientovaná cesta z x do y nebo orientovaná cesta z y do x (nebo obě). Navrhněte algoritmus na testování polosouvislosti grafů, který poběží v $O(n+m)$, kde n je počet vrcholů a m počet hran v grafu G.

Řešení Ukážeme algoritmy i pro horší časové složitosti:

1. V čase $O(n^2(n+m))$

   Stačí spustit[4] pro každé dva vrcholy grafu prohledávání do hloubky (DFS).

2. V čase $O(n(n+m))$

   Nejdříve spočítáme matici dosažitelnosti tím, že spustíme n krát DFS (tedy $O(n(n+m))$). Pak zjistíme prohlédáním celé matice dosažitelnosti, jestli existují indexy i, j takové, že

   $$D[i,j] = D[j,i] = 0.$$

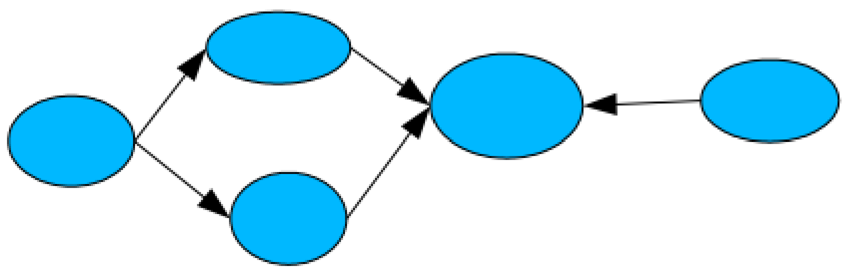
   Toto stihneme v čase $O(n^2)$. Dohromady máme $O(n(n+m))$.

3. V čase $O(n+m)$

   (a) Nejdříve najdeme v grafu G silně souvislé komponenty $S_1, \dots S_p$, což stihneme v čase $O(n+m)$. Algoritmus na hledání silně souvislých komponent:

[4] Časová složitost DFS je $O(n+n)$.

```
STRONGLY-CONNECTED-COMPONENTS (graph G)
1  call DFS (G) to compute finishing times f[u]
       for each vertex u
2  compute G^T (graph G with reversed edges)
3  call DFS (G^T), but in the main loop of DFS, consider
       the vertices in order of decreasing f[u]
       (as computed in line 1)
4  output the vertices of each tree in the depth-first
       forest formed in line 3 as a separate strongly
       connected component
```



Obrázek 3: Silně souvislé komponenty

Pozorování: Žádné nalezené silně souvislé komponenty neleží na kružnici. Jinak by tyto silně souvislé komponenty spolu s kružnicí tvořili novou silně souvislou komponentu, která je ostře větší než SSK, ze kterých by se skládala.

(b) Silně souvislé komponenty bychom mohli uspořádat pomocí topologického třídění[5]. Algoritmus pro topologické třídění:

```
TOPOLOGICAL-SORT(graph G)
1  call DFS(G) to compute finishing times f[v]
     for each vertex v
2  as each vertex is finished, insert it onto
     the front of a linked list
3  return the linked list of vertices
```

[5] Místo vrcholů bychom třídíli silně souvislé komponenty. Mohli bychom algoritmus použít díky pozorování.

Pokud se však podíváme[6] na algoritmus pro STRONGLY-CONNECTED-COMPONENTS, tak druhý DFS průchod (řádek 3) vrací silně souvislé komponenty v topologickém uspořádání a proto TOPOLOGICAL-SORT aplikovat znovu nepotřebujeme.

(c) Abychom vyřešili úlohu, tak algoritmus STRONGLY-CONNECTED-COMPONENTS z bodu (a) doplníme na řádce 3 o uložení informace, že při stavění i-tého stromu S_i v grafu G^T (což je topologicky i-tá SSK grafu G, jak bylo uvedeno v bodě (b)) vede z nějakého vrcholu $u \in S_i$ hrana do nějakého vrcholu $v \in S_{i-1}$. Tím pádem vede hrana v grafu G ze SSK S_{i-1} do SSK S_i. G je polosouvislý tehdy a jen tehdy když tohle platí pro každé i.

3. cvičení

Příklad 1 Nechť máme k dispozici „černou skřínku", která umí řešit SAT (rozhodovací problém splnitelnosti CNF formulí) v polynomiálním čase. Skřínka odpovídá pouze ANO-NE. Zkonstruujte algoritmus, který pro danou CNF najde v polynomiálním čase (libovolné) splňující ohodnocení proměnných, pokud takové ohodnocení existuje.

Poznámky

- Formule je ve tvaru *konjunktivní normální formy (CNF)*, jestliže je ve tvaru konjunkce klauzulí (disjunkce literálů), tedy např. formule:

  $(A \vee B) \wedge (\neg B \vee C \vee \neg D) \wedge (D \vee \neg E)$

  je ve tvaru CNF.

- Zvláštním případem je prázdná formule, ta je z definice nesplnitelná.

Řešení (algoritmus)

1. Pusť blackbox na vstupní formuli a ...

   (a) pokud vrátí NE, pak SKONČI, formule nelze splnit.

   (b) pokud vrátí ANO, pak přejdi na krok 2)

[6] Nebo si přečtete kapitolu 22 a článek o SSK v *Introduction to Algorithms, 2ed*

2. Vyber nějakou proměnnou x z formule a jdi na krok 2a)

   (a) dosaď $x = 0$ do aktuálního CNF a pusť blackbox ...

      - pokud blackbox vrátí ANO, tak přejdi na krok 2)
        [Nepokazili jsme volbou $x = 0$ splnitelnost, tak můžeme pokračovat na dalších proměnných]
      - pokud vrátí NE, tak vrať dosazení $x = 0$ a přejdi na krok 2b)
        [Pokazili jsme splnitelnost volbou $x = 0$, tak tento krok vrátíme a již s jistotou v kroku 2b) můžeme dosadit $x = 1$]

   (b) dosaď $x = 1$ a jdi na krok 2)

Poznámky

- Do kroku 2 se dostanu pouze pokud formule je splnitelná.
- Místo dosazování hodnot za proměnné bych mohl do formule přidávat *literály* (proměnná nebo negace proměnné) a tím vynucovat jejich hodnotu. Krok 2a bych tedy mohl nahradit tím, že na konec formule přidám $\wedge \neg x$.

Složitost algoritmu: $O(n)$

Řešení (slovně) Postupně se dosazují hodnoty za jednu proměnnou, pokud volba projde (skříňka řekne ANO), pak dosadím za další proměnnou a postup opakujeme; pokud skříňka řekne ne, pak dosadíme opačnou hodnotu (tj. pokud jsme prve dosadili 1, tak teď dosadíme 0) a opět pošleme do skříňky (pokud mi to pro 0 (resp. 1) nevyšlo, pak pro 1 (resp. 0) to už vyjít musí (jinak by neexistovalo žádné ohodnocení)).

Příklad 2 Problém vrcholového pokrytí (VP).

- Definice na úvod: Nechť $G = (V, E)$ je neorientovaný graf, nechť $S \subseteq V$, pak S je *vrcholové pokrytí* G, jestliže $\forall e \in E$ platí e je incidentní[7] s S.

  [Tedy: S je vrcholové pokrytí G, pokud pro každou hranu $(u, v) \in E$ platí že, $u \in S$ nebo $v \in S$.]

[7] Jestliže $(u, v) \in E$, pak vrcholy u a v jsou incidentní s hranou e a žádné jiné vrcholy nejsou.

- Vstup: neorientovaný graf $G = (V, E)$ a $k \in N$

Mějme blackbox, který dokáže o daném grafu říci, zda existuje v G vrcholové pokrytí, které je $\leq k$. Navrhněte algoritmus, který pomocí blackboxu v polynomiálním čase zkonstruuje pro daný graf G jeho minimální vrcholové pokrytí.

Poznámka Pokud má graf vrcholové pokrytí velikosti k, pak jistě existují vrcholová pokrytí velikosti $k + 1, k + 2, \ldots, |V|$, naopak to obecně neplatí.

Řešení

1. Pomocí nejvýše $|V| = n$ dotazů na blackbox získáme správnou hodnotu parametru k. Ptáme se blackboxu postupně zda existuje vrcholové pokrytí velikosti 0,1, ... dokud blackbox neodpoví, že takové vrcholové pokrytí existuje.

   Ve skutečnosti to umíme rychleji. Stačí využít techniku půlení intervalů a pak nám stačí $\lceil log_2 n \rceil$ dotazů na blackbox.

   [$S = V$ je jistě vrcholové pokrytí, tedy nejpozději na n se zastavíme.]

2. Algoritmus:
   (a) Označíme všechny vrcholy bílou barvou.
   (b) Dokud existuje bílý vrchol, tak vyberu náhodně nějaký bílý vrchol v, označím ho červeně a zeptám se, zda na množině bílých vrcholů existuje vrcholové pokrytí o velikosti $k - 1$:
      i. pokud existuje, pak odeberu vrchol v (čímž odstraním i hrany s ním incidentní, ty mám již ale pokryté vrcholem v) a nastavím $k = k - 1$.
      ii. pokud neexistuje, tak označím v černě (tím říkám, že $v \notin S$) a k neměním.

Složitost

- Zjištění k: $O(log_2 n)$ použití BB
- Počet vrcholů navštívených ve smyčce je nejvýše n.

Dohromady tedy $O(log_2 n + n) = O(n)$ použití BB.

Poznámka Člověk by si mohl říct, že algoritmus upraví tak, že odebere vrchol v i v případě ii). To však rozbije algoritmus, protože se s v mohou odeberat i nějaké hrany, které již později nemusíme pokrýt.

Příklad 3 Problém TAUT.

- Vstup: Booleovská formule F sestávající z proměnných $x_1, x_2, \ldots, x_n$ a operací $\{\wedge, \vee, \neg, (,)\}$
- Otázka: Je F tautologie?

1. TAUT $\in$ NP
2. TAUT $\in$ CO-NP
3. Jaká je složitost TAUT, pokud je F ve tvaru CNF?
4. Jaká je složitost TAUT, pokud je F ve tvaru DNF?

Řešení

1. Neví se. Intuitivně je jasné, že certifikát se bude hledat obtížně - jak najít certifikát, že formule ve všech ohodnoceních platí?

   Pokud však TAUT $\in$ NP, pak NP $\equiv$ co-NP. Proč?

2. Ano. Pokud máme formuli, která není tautologií, tak stačí najít jedno ohodnocení proměnných (certifikát) takových, že formule není pravdivá. Takový certifikát jde zkontrolovat v polynomiálním čase, takže máme splněny všechny požadavky třídy co-NP.

3. Lineární. Na formuli "útočíme po klauzulích". Stačí aby se v jedné klauzuli nevyskytovala ani jedna proměnná $x_1, \cdots, x_n$ jako komplementární pár[8] a celá formule není tautologie. Aby tedy byla formule tautologie, tak se musí v každé klauzuli vyskytovat alespoň jedna proměnná jako komplementární pár.

   Příklad:

   $$(A \vee B) \wedge (\neg B \vee C \vee \neg D)$$

   zde stačí zvolit ohodnocení $v(A) = 0$ a $v(B) = 0$ (a libovolné ohodnocení ostatních výrokových proměnných) a formule není pravdivá.

[8] Tedy ve formě gace a zároveň ve formě negace.

4. Na $\overline{TAUT}$ můžeme převést problém SAT. Stačí v $\overline{TAUT}$ znegovat celou formuli a dostaneme formuli v CNF, ve které nehledáme vyvracející ohodnocení, ale splňující. TAUT je tedy co-NP-úplný[9].

4. cvičení

Příklad 1 Definujme problém LOUP (loupežníci) následovně:
Instance: Přirozená čísla $a_1, \dots, a_n$
Otázka: Existuje podmnožina T množiny indexů $S = \{1, \dots, n\}$ taková, že:

$$\sum_{i \in T} a_i = \sum_{i \in S \setminus T} a_i$$

Dokažte, že LOUP je NP-úplný problém.

Řešení Dokazujeme, že LOUP je NP-úplný problém.

1. LOUP $\in$ NP

   - Certifikát: Množina T.

2. Je LOUP NP-těžký?

   Převedeme na problém *součtu podmnožiny*[10] s tím, že součet je $b := \frac{\sum a_i}{2}$ (dělíme na půlky). Tedy:

   SP $\propto$ LOUP

$$a_1, a_2, \dots, a_n, b \tag{1}$$

$$a_1, a_2, \dots, a_n, b^* = \underbrace{\sum_{i=1}^{n} a_i}_{\text{ozn. } A} - b \tag{2}$$

   Je zřejmé, že (1) má řešení právě tehdy když (2) má řešení[11]. Nyní zadefinujeme b' takto:

[9] co-SAT je kontradikce CNF.
[10] Existuje pro $a_1, \dots, a_n \in \mathcal{Z}^+$ množina indexů $S \subseteq \{1, \dots, n\}$ taková, že $\sum_{i \in S} a_i = b$?
[11] Jinými slovy: Je jedno, jestli hledáme součet 700,- nebo doplněk 300,- v 1000,-

- jestliže $b \geq \frac{1}{2}A$, pak $b' = b$,
- jestliže $b < \frac{1}{2}A$, pak $b' = A - b$.

Proměnnou b' zavádíme proto, abychom nepracovali s příliš malým b. Dodefinujeme ještě prvek a_{n+1} takto:

$$a_{n+1} := b' - (A - b') = 2b' - A,$$

což odpovídá doplnění jednoho předmětu tak, aby součet všech předmětů byl $2b$ (b je hledaná půlka).

Věta: Zkonstruovaná instance LOUP: $a_1, \dots, a_n, a_{n+1}$ má řešení $\Leftrightarrow$ vstupní instance SP: $\{a_1, \dots, a_n\}, b'$ má řešení.

Důkaz.

”$\Rightarrow$” Nechť LOUP($\{a_1, ..a_n\} \cup \{a_{n+1}\}$) má řešení $I \subseteq \{1, 2, \dots, n+1\}$. Pak:

$$\sum_{i \in I} a_i = \sum_{j \in \{1,2,\dots,n+1\} \setminus I} a_j$$

$$\sum_{i=1}^{n+1} a_i = A + 2b' - A = 2 \cdot b'$$

Jestliže $n + 1 \in I$, pak $\{1, 2, \dots, n + 1\} \setminus I$ je řešením SP, stejný argument platí pro $n + 1 \in \{1, 2, \dots, n + 1\} \setminus I$.

”$\Leftarrow$” Předpokládáme, že součet podmnožiny má řešení $I \subseteq S$, pak platí $\sum_{i \in I} a_i = b$. Mohou nastat dvě možnosti:

(a) $b' = b$ pak $I' = I$

(b) $b' = A - b$ pak $I' = S \setminus I$

Z čehož dostáváme, že pro I' platí:

$$\sum_{i \in I'} a_i = b'.$$

Dále platí:

$$\sum_{i \in S \setminus I'} a_i + a_{n+1} = A - b' + \underbrace{2b' - A}_{a_{n+1}} = b'$$

I' je tedy řešením instance LOUP.

□

Příklad 2 Nejkratší cestu mezi dvěma vrcholy v neorientovaném váženém grafu lze nalézt pomocí Dijkstrova algoritmu, pokud jsou zadané váhy na hranách nezáporné (cestou rozumíme sled hran BEZ opakování vrcholů). Jak se změní složitost této úlohy, pokud povolíme, aby byly váhy záporné?

Řešení Problém je NP-úplný.

Důkaz.

1. PATH $\in$ NP (certifikátem je nejkratší cesta).
2. Převedeme HK $\propto$ HC[12] a HC $\propto$ PATH

   HK $\propto$ HC: V grafu zvolím vrchol x a zkopíruji jej do nového vrcholu y (hrany vedoucí do x povedou ze stejných vrcholů také do y)

   ```
   o  o  o          o  o  o
    \ | /      ->     x  x      (uplny bipart. gr.)
      o              o   o
                     x   y
   ```

   Nyní hledám v novém grafu HC z x do y. Jestliže existuje, vede v původním grafu HC z x do v, kde v je předposlední vrchol v HC v novém grafu. Tím pádem je ve starém grafu i HK (mezi x a v vede hrana).

   HC $\propto$ PATH: Ohodnotím všechny hrany -1 a hledám nejkratší cestu. Má-li velikost -(n-1), prošla všemi vrcholy a je to tedy HC. Kratší být nemohla, protože se vrcholy nesmí opakovat a všechny jsou vyčerpány.

□

Příklad 3 Definujme problém 0-1 CP (celočíselné programování) následovně:
Instance: Celočíselná matice A řádu m krát n a celočís. vektor b délky m
Otázka: Existuje vektor x délky n obsahující pouze čísla 0 a 1 takový, že $Ax \leq b$?
Dokažte, že 0-1 CP je NP-úplný problém.

[12] Hamiltonovská cesta: Existuje v grafu cesta, která prochází všemi vrcholy?

Řešení Integer linear programming (ILP) is like linear programming, with the additional constraint that all variables must take on integral values. The decision version of integer programming asks whether or not there exists a point satisfying all the constraints (for the decision version there is no objective function).

Tvrzení: ILP is NP-complete.

Důkaz. 1. ILP is in NP.

2. We can reduce 3SAT to ILP:

Let the variables in the 3SAT formula be $x_1, x_2, ..., x_n$. We will have corresponding variables $z_1, z_2, ..., z_n$ in our integer linear program. First, restrict each variable to be 0 or 1:

$$\forall i : 0 \leq z_i \leq 1$$

Assigning $z_i = 1$ in the integer program represents setting $x_i = true$ in the formula, and assigning $z_i = 0$ represents setting $x_i = false$.

For each clause like $(x_1 \vee \neg x_2 \vee x_3)$, have a constraint like:

$$z_1 + (1 - z_2) + z_3 > 0.$$

To satisfy this inequality we must either set $z_1 = 1$ or $z_2 = 0$ or $z_3 = 1$, which means we either set $x_1 = true$ or $x_2 = false$ or $x_3 = true$ in the corresponding truth assignment. □

By Jindra:

Tvrzení: 0-1 CP je NP-úplný problém.

Řešení SAT $\propto$ 0-1 CP.

Mějme formuli v CNF. Klauzule převedeme na nerovnosti následujícím způsobem: Z operací disjunkce uděláme operace sčítání. Proměnné A_i v negaci zaměníme za rozdíl $1 - A_i$ a ostatní proměnné ponecháme. O tomto výrazu prohlásíme, že je ≥ 1. Potom je nerovnost splněna právě tehdy když je splněna i klauzule.

Příklad:

$$(a_1 \vee \neg a_2 \vee a_3) \Leftrightarrow a_1 + (1 - a_2) + a_3 \geq 1$$

Předpokládáme, že ohodnocení proměnných nabývá hodnot 0,1.

Celá formule je potom soustavou nerovnic. Každou nerovnost ještě vynásobíme (-1), aby se změnilo znaménko, a upravíme. Koeficenty napíšeme do matice A a b je vektor pravých stran nerovností.

Příklad:

$$a_1 + (1 - a_2) + a_3 \geq 1$$
$$\Leftrightarrow -a_1 - 1 + a_2 - a_3 \leq -1$$
$$\Leftrightarrow -a_1 + a_2 - a_3 \leq 0$$

Důkaz. CP $\Rightarrow$ SAT

Pokud CP získalo vektor x, pak ji použijeme jako ohodnocující vektor proměnných. Jelikož jsou splněny všechny nerovnosti, jsou splněny i klauzule.

SAT $\Rightarrow$ CP

Existuje-li ohodnocení proměnných, dá se z něj získat vektor, který je výsledkem CP. $\square$

Příklad 4 Definujme problém IP (izomorfismus podgrafů) následovně:
Instance: Neorientované grafy G a H
Otázka: Je graf G izomorfní s nějakým podgrafem grafu H? (varianta 1)
Otázka: Je graf G izomorfní s nějakým indukovaným podgrafem grafu H? (varianta 2)
Dokažte, že obě varianty IP jsou NP-úplné problémy.

Řešení indukované varianty (varianta 2) [13]

The subgraph-isomorphism problem takes two graphs G_1 and G_2 and asks whether G_1 is isomorphic to a subgraph of G_2. (Two graphs are isomorphic, if there is a permutation of the verteces which transform one graph into the other, preserving the edges). Prove that the subgraph isomorphism problem is NP-complete, using NP-complete problems discussed in class.

Důkaz. First we prove that the subgraph-isomorphism problem is in NP. The certificate is $(G_1 = (V_1; E_1); G_2 = (V_2; E_2); \phi : V_1 \to V_2)$. The verifying algorithm checks if ϕ isa one-to-one function, and for all $u, v \in V_1$ whether $(u; v) \in E_1$ if and only if $(\phi(u); \phi(v)) \in E_2$.

Secondly, we prove that CLIQUE $\propto$ SUBGRAPH ISOMORPHISM. Let $(G = (V; E); k)$ be an input instance for CLIQUE. Define G_1 to be the

[13] Lze v nezměněné podobě použít i na variantu 1

complete graph on k vertices, and G_2 to be the graph G. Then $(G1; G2) \in$ SUBGRAPH ISOMORPHISM if and only if $(G; k) \in$ CLIQUE. □

Poznámka: Graf G ze zadání příkladu je v důkazu výše uplný graf G_1 na k vrcholech a H odpovídá grafu G_2.

Poznámka: Proč je toto řešení indukované varianty? Protože, pro všechny vrcholy, které netvoří kliku (které odebereme z H) se odeberou i hrany s těmito vrcholy incidentní.

Řešení neindukované varianty (varianta 1)

Důkaz. Nechť G_1 je instance problému Hamiltonovské kružnice na grafu G s n vrcholy. Nechť $G_2 = C_n$ je prostá kružnice na n vrcholech. Pokud G_1 obsahuje HK, tak je v něm zároveň i podgraf izomorfní s G_2. Pokud G_1 neobsahuje HK, tak neexistuje izomorfismus mezi G_1 a G_2.

□

5. cvičení

Příklad 1 Nechť máme k dispozici „černou skřínku“, která umí řešit rozhodovací verzi problému součtu podmnožiny v polynomiálním čase. Skřínka odpovídá pouze ANO-NE. Zkonstruujte algoritmus, který pro daný vstup optimalizační verze problému součtu podmnožiny najde v polynomiálním čase (vzhledem k délce binárního zápisu vstupních dat) optimální řešení.

Formální zápis úlohy:
Vstup: Čísla $x_1, \ldots, x_n, t \in \mathbb{Z}^+$.
Výstup: Množina $S \subseteq \{1, \ldots, n\}$ taková, že $\sum_{i \in S} x_i \leq t$ a $\sum_{i \in S} x_i$ je maximální.

Řešení

Definujme čísla: $y_0, \ldots, y_l$, kde $y_i = 2^i$ a $l = \lfloor log_2 t \rfloor$. Je důležité si uvědomit, že platí

$$\sum_{i=0}^{n-1} 2^i = 2^n - 1,$$

což je základní vlastnost binárních čísel.

Algoritmus

1. test: čísla $x_1, \ldots, x_n, y_0, \ldots, y_{l-1}$ a požadovaný součet t. Jestliže, je výsledek testu:

- ANO - pak hledané maximum leží v intervalu $[t - (2^l - 1), t]$.
- NE - pak hledané maximum leží v intervalu $[0, t - (2^l - 1)]$

Odebereme y_{l-1} a iterujeme další testy.

By Vrtule:

```
BOOL ret = test: BB ({x_1, ..., x_n, y_0, ..., y_l-1}, t)
if (ret)
// Maximum se nachazi v intervalu (t - (2^l - 1) ; t)
else {
// maximum se nachazi v intervalu (0 ; t - (2^l - 1))
t = t - y_l-1;
}
l--;
```

Ono totiz. Dostavas-li od BB odpovedi ANO, tak se ti cilovy interval stale zmensuje. Kdyz dostanes odpoved NE, tak vis, ze se maximum nenachazi uz blize k t. Takze dostanes urcity interval (zalezi na poctu dotazu ANO, nez prisel NE), kde to maximum je. Odpoved NE pro konkretni l ti rekne, ze neni podstatne, zda v mnozine $y_l - 1$ je ci neni. At uz s nim, nebo bez nej, soucet hodnoty t proste nedosahne. Takze podle o nej muzes snizit t-ckova jit do dalsi iterace algoritmu.

Zkus si to nakreslit. Ten interval se bude stale pulit imho. A jakmile ziskas presnou hodnotu maxima, tak uz je to jednoduche. Provedes n dotazu do BB, cimz zjistis, zda dane x_i se v tom souctu nachazi, nebo ne:

Odeberes x_i z mnoziny a zeptas se BB:

ANO — x_i neni v souctu

NE — x_i je v souctu.

Jen ted nevim, zda jsou tyhle kroky zbytecne, nebo ne. Kazdopadne to znamena $n + logt$ dotazu na BB, coz je polynomialni vzhledem ke vstupu $(n * logt + logt + logt * logt)$, coz bude asi vse $O(n)$.

Příklad: Nechť $t = 18$ a dále

$$\begin{aligned} x_1 &= 1 & y_0 &= 2^0 = 1 \\ x_2 &= 3 & y_1 &= 2^1 = 2 \\ x_3 &= 5 & y_2 &= 2^2 = 4 \\ x_4 &= 8 & y_3 &= 2^3 = 8 \\ & & y_4 &= 2^4 = 16 \end{aligned}$$

1. test: $x_1, \ldots, x_4, y_0, \ldots, y_3$, $l = 4$
Odpověď BB je ANO, řešení je v intervalu:
$[t - (2^l - 1), t] = [18 - 15, 18] = [3, 18]$
2. test: $x_1, \ldots, x_4, y_0, \ldots, y_2$, $l = 3$
Odpověď BB je ANO, řešení je v intervalu:
$[18 - (2^3 - 1), 18] = [11, 18]$
3. test: $x_1, \ldots, x_4, y_0, y_1$, $l = 2$
Odpověď BB je ANO, řešení je v intervalu:
$[18 - (2^2 - 1), 18] = [15, 18]$
4. test: $x_1, \ldots, x_4, y_0$, $l = 1$
Odpověď BB je ANO, řešení je v intervalu:
$[18 - (2^1 - 1), 18] = [17, 18]$
5. test: $x_1, \ldots, x_4$, $l = 0$
Odpověď BB je NE, řešení je v intervalu:
$[17, 18 - (2^0 - 1)] = [17, 18]$
Řešením je tedy 17.

Příklad 2 Popište jak lze pro libovolné dané n zkonstruovat graf na n vrcholech takový, že aproximační algoritmus pro vrcholové pokrytí s poměrovou chybou $r = 2$ (prezentovaný na přednášce) na tomto grafu vrací pokrytí právě dvakrát větší než je velikost optimálního vrcholového pokrytí (tj. ukažte, že dokázaný odhad poměrové chyby je těsný).

Řešení

- První řešení:

```
o   o   o
|   |   |    . . .
o   o   o
```

- Budeme-li požadovat navíc souvislý graf:

```
    o ---          <- vrchol stupne k
   / \   \    ...
  o   o    o
 / \ / \  / \  ...
 o o o o  o o
```

  Nebo na úplném bipartitním grafu. Společná vlastnost těchto dvou příkladů je ta, že obsahují nezávislou množinu.

- Graf "hvězda"- algoritmus vybere hranu, ale stačil by střed, tedy poměrová chyba je 2.

Příklad Navrhněte algoritmus, který v polynomiálním čase nalezne optimální VP pro les/strom.

Řešení Idea: Je zbytečné do VP dávat listy.

- Pro každý list bude jeho rodič ve VP
- Odeberu hrany incidentní s nově označenými vrcholy.
- Vznikne nový les. Udělám to samé, dokud nedojdou hrany.

6. cvičení

Příklad 1 Bottleneck TSP: vstupem je úplný ohodnocený neorientovaný graf s nezápornými váhami na hranách (stejně jako u obyčejného TSP), o kterých navíc předpokládáme, že splňují trojúhelníkovou nerovnost. Úkolem je opět najít nejkratší Hamiltonovskou kružnici ve vstupním grafu, ovšem délka kružnice není v tomto případě rovna součtu délek hran na kružnici, ale maximu z délek hran na kružnici. Nejdříve dokažte, že Bottleneck TSP je NP-těžký a poté navrhněte polynomiální aproximační algoritmus s poměrovou chybou r=3.

Řešení Nejdříve dokážeme, že BTSP je NP-úplný.

1. HK $\propto$ BTSP: Zadání HK: $G = (V, E)$

   Sestrojím graf $G^H = (V, V \times V)$ zúplnění grafu G a definuji:

   $w(e) = 1 \ldots$ pro $e \in E$,

   $w(e) = 2 \ldots$ pro $e \notin E$

   ohodnocení hran.

   Pokud BTSP najde HK s váhou 1, existuje v G HK a naopak.

2. Aproximační algortmus:

   - Zjistit minimální kostru T grafu G.
   - V T^3 existuje[14] hamiltonovská kružnice (T byl souvislý) $\rightarrow$ najdu tuto HK.
   - Tato HK je řešení BTSP s $r \leq 3$.

   Důkaz. **Fakt 1:** Váha optimální HK $f(HK)$ je větší rovno než váha minimální kostry T grafu G (označme ji $f(t)$)

   Důkaz:

   Sporem. $f(HK_{OPT}) < f(T)$. Potom $\exists e \in T$ tak, že $w(e) >$ váha nejtěžší hrany v HK_{OPT}. Tedy lehčí hrany vedoucí mezi množinou A a B (spojené hranou e). Existují A a B a tudíž T není minimální kostra.

   Fakt 2: HK v T^3 (která existuje) má díky trojúhelníkové nerovnosti velikost max. 3x větší než T.

   Důkaz:

   Vezmeme hranu e nejtěžší v T. V T^3 není žádná hrana nad víc, než nad třemi hranami z T.

   ```
       t
   /----------\
   o---o---o---o
     e1  e2  e3
   ```

[14] Toto je třeba dokázat: Nechť mám kostru T grafu G, pak v grafu T^3 existuje hamiltonovská kružnice. Důkaz se dělá indukcí podle počtu vrcholů.

Kde $e_1, e_2, e_3, \ldots$ jsou hrany z T.

$w(e_1, e_2, e_3) \leq w(e) = f(T)$

Díky troj. nerovnosti $w(t) \leq w(e_1) + w(e_1) + w(e_3) \leq 3 \cdot f(T) \Rightarrow$ $f(APR) \underbrace{\leq}_{F2} f(T) \underbrace{\leq}_{F1} \leq 3 \cdot f(OPT)$

□

Příklad Bin-packing. Čísla $a_1, \ldots, a_n$ $(\forall i : 0 < a_i < 1)$ se mají rozdělit do co nejmenšho počtu binů velikosti:

1. Dokažte, že bin-packing je NP-těžký
2. Dokažte, že apr. algoritmus FIRST-FIT (dá to do prvního, kam se to vejde), má poměrovou chybu nevýše 2.

Řešení

1. NP-těžký: LOUP $\propto$ BP

   Úloha $\ldots a_1, \ldots, a_n$ lze rozdělit napůl?

   Úloha pro bin-packing:

   čísla $b_1, \ldots, b_n$ pro která platí $b_i = \frac{2a}{\sum_{j=1} a_j}$ (kdyby $\exists i a_i > \frac{\sum_{j=1} a_j}{2}$ tak to nejde rozdělit) mám rozdělit do dvou binů. Pokud to jde v binech jsou rozděleny $a_1, \ldots, a_n$ napůl (přeškálovaně)

2. FF: Označme Opt = optimální počet binů, $A = \sum_{i=1} a_i$, f = výsledek algoritmu FF (počet binů)

   Platí: FF má nejvýše jeden bin zaplněný méně jak z poloviny

   Dk: Sporem. Kdyby měl zaplněné méně jak z poloviny 2, tak se tyto dva biny dají slít, což je spor s použitím FIRST-FIT algoritmu.

   Platí: $Opt \leq 2\lceil A \rceil$

   Dk: Sporem. Předpokládejme, že $f > Opt\,(\geq 2\lceil A \rceil) \Rightarrow f > \lceil A \rceil \Rightarrow f \geq 2\lceil A \rceil + 1$.

   f má ale zaplněný nejvýše 1 bin méně než z poloviny, a tudíž má alespoň $2\lceil A \rceil$ binů zaplněno více jak z poloviny, takže tam algoritmus naskládal víc věcí, než bylo v zadání.

Zdroje

- http://www.cs.cmu.edu/ avrim/451/recitations/rec0408.txt
- Příklady ve studnici, příklady na foru, poznámky