Složitost I - cvičení

Martin Všetička

1 Cvičení 2010-10-04

1.1 Příklad

Def $S = \{1, \dots, n\}$ 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.

Úkol Na tomto cvičení se hledají největší (v kardinalitě) množiny kompatibilních úkolů. Používá se 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.

Možná kritéria pro algoritmus

1. Nejmenší hodnota 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!).

2. 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.

3. 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.

4. 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.

1.2 Příklad

Úkol Najděte rozklad všech úkolů na co nejmenší počet kompatibilních množin.

Poznámka Zadání si můžeme představit tak, že máme učebny a chceme přednáškami zaplnit co nejméně učeben.

Řešení Jak by nás napadlo, že to jde dělat:

- Iterativní použití kritéria 4 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 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 algoritmus, vybírajícího maximální kompatibilní množiny.

  Protipříklad:

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

- Iterativní použití kritéria 3 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 $\forall j \in \{1, \ldots, k-1\}$ 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.

  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 úplný graf K_4 nejde. Rozvrhy odpovídají pouze jedné skupině grafů, kterým se říká *intervalové* a u nich lze najít chromatické číslo rychle.

2 Matroidy

2.1 Příklad

Nechť S je konečná a neprázdná. Nechť $I_k = \{A \subseteq S \mid |A| \leq k\}$. Dokažte, že (S, I_k) je matroid $\forall k$.

Řešení

1. S konečná, neprázdná – ANO
2. Dědičná vlastnost je zřejmá přímo z definice I_k. Jestliže mám $B \subseteq S$, pak $A \subseteq B$ má velikost určitě $\leq k$.
3. 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.

2.2 Příklad

- 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 $I = \{A \subseteq S \mid \forall i : |A \cap S_i| \leq 1\}$.

Dokažte, že (S, I) je matroid.

Řešení

1. S konečná, neprázdná – ANO
2. Dědičná vlastnost platí. Jestliže mám $B \in I$, pak pro $A \subseteq B$ určitě platí $\forall i : |A \cap S_i| \leq 1\}$ – ANO.
3. 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.

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

2.3 Příklad (DÚ)

Nechť $M = (S, I)$ je matroid. Dokažte, že $M' = (S, I')$ je také matroid, kde $I' = \{A \subseteq S \mid$ taková, že $S \setminus A$ obsahuje nějakou maximální nezávislou množinu z $I\}$

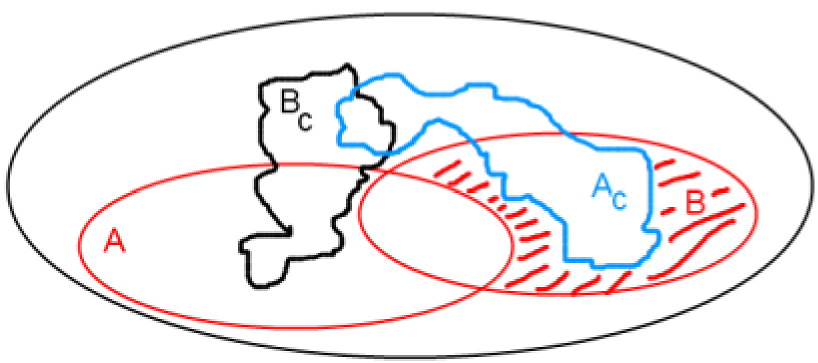
Řešení

1. S je neprázdná, protože M je matroid.
2. 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]
3. Výměnná vlastnost:

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

   (a) 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.

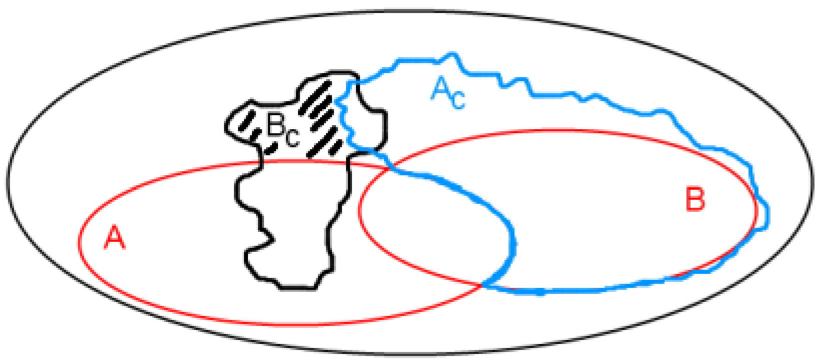
   (b) Neexistuje prvek x z případu a), to znamená že platí $B \setminus A \subseteq A_C$.

👁 **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 $|A \setminus (A \cap B)| < |B \setminus (A \cap B)|$ a z dále plyne, 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.

$\square$



- 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.

3 Cvičení 2010-10-18

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 $outdegree(x) = 0$,

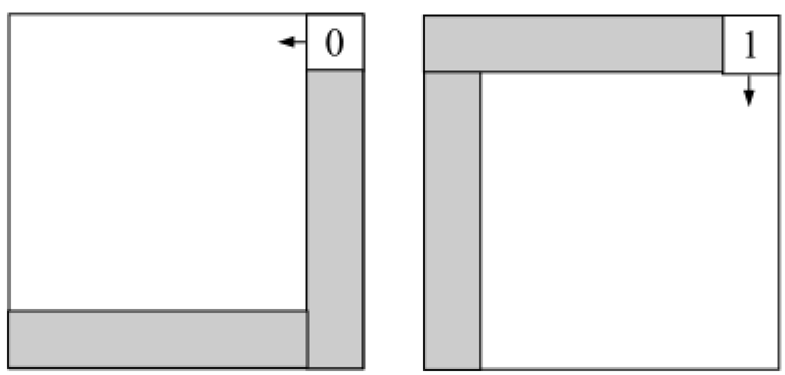
přičemž algoritmus smí použít (přečíst) pouze O(n) prvků matice. Předpokládejme, že před zahájením algoritmu je již celá matice načtena do paměti.

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 data 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[1] 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?

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

- 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 $outdegree(i) \geq 1$.

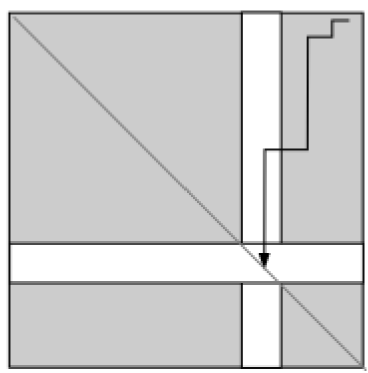
V každém kroku procházky tedy dokážeme vyloučit jeden vrchol, který nemůže být stokem. Stačí tedy projít $n - 1$ pozic v matici A a o posledním vrcholu rozhodnout, zda je nebo není stokem, jelikož je to jediný kandidát.

Po $n-1$ krocích skončíme na diagonále - proč? Uděláme z pravého horního rohu matice D kroků dolů a L doleva[2]. 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 jediného kandidáta na stok a proto ho ověříme (tj. zda bílý řádek tvoří samé nuly a bílý sloupec samé jedničky (kromě prvku na diagonále)).

Složitost aloritmu:

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

[2]Platí, že $D + L = n - 1$.

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[3] 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

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

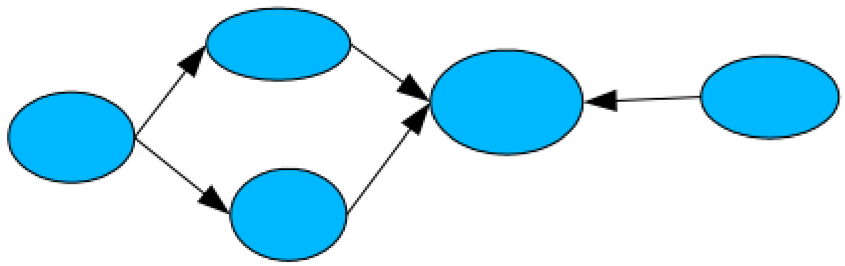
   Toto stihneme v čase $O(n^2)$. Dohromady máme $O(n(n+m))$.

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:

   ```
   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
   ```

[3] Časová složitost DFS je $O(n+n)$.



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[4]. 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
```

Pokud se však podíváme[5] 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 $v \in S_{i-1}$.

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

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

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.

4 Cvičení 2010-11-08

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)
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 $\wedge\neg x$.

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[6] s S.

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

- 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.

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

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.

   [$S = V$ 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.

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. Pokud ANO, pak NP $\equiv$ Co-NP. TODO
2. Okamžitě: Ano. Certifikát pro zápornou odpověď: Ohodnocení při němž formule není pravdivá.

   Potřebujeme algoritmus, který to v polynomiálním čase umí.
3. Na formuli "útočíme po klauzulích". Stačí aby se v jedné klauzuli nevyskytovala nějaká proměnná $x_1, \cdots, x_n$ jako komplementární pár[7] a celá formule není tautologie. Pokud se proměnná nevyskytuje v klauzuli vůbec, tak to samozřejmě nevadí.

   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.

   Lze v lineárním čase otestovat. TODO JAK PRESNE?
4. Pozorování: F je nesplnitelná $\Leftrightarrow$ $\neg F$ je nesplnitelná. TODO

5 UNSORTED

Příklad 1 Definujme problém LOUP (loupežníci) následovně:
Instance: Přirozená čísla $a_1, \ldots, a_n$
Otázka: Existuje podmnožina T množiny indexů $S = \{1, \ldots, n\}$ 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 j NP-úplný problém.

1. LOUP $\in$ NP

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

- Převedeme na problém *součtu podmnožiny*[8] s tím, že součet je $b := \frac{\sum a_i}{2}$ (dělíme půlky)
- Certifikát: Množina T.

2. Je LOUP NP-těžký?

   SP [znak redukce] LOUP

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

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

Je zřejmé, že (1) má řešení právě tehdy když (2) má řešení[9]. Nyní zadefinujeme b' takto:

- 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: $a_1, \ldots, a_n, a_{n+1}, b'$ má řešení $\Leftrightarrow$ vstupní instance SP: $\{a_1, \ldots, a_n\}, b$ má řešení.

Důkaz.

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

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

$$\sum_{i \in I} a_i = (A + 2b' - A)/2 = b' = \sum_{j \in \{1,2,\ldots,n+1\} \setminus I} a_j$$

[8] Existuje pro $a_1, \ldots, a_n \in \mathcal{Z}^+$ množina indexů $S \subseteq \{1, \ldots, n\}$ taková, že $\sum_{i \in S} a_i = b$?

[9] Jinými slovy: Je jedno, jestli hledáme součet 700,- nebo doplněk 300,- v 1000,-

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

"$\Leftarrow$" Předpokládáme, že součet podmnožiny má řešení $I \subseteq S$ (tj. množina indexů množiny K), 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:

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

I' je tedy řešením instance LOUP.

□