

Složitost I - cvičení

Martin Všeticka

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 $V(I, X) = \text{YES}$.
- If I is a NO-instance, then for all X , $V(I, X) = \text{NO}$.

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

co-NP: Problems where 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 , $V(I, X) = \text{YES}$.
- If I is a NO-instance, then there exists X such that $V(I, X) = \text{NO}$.

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 use it to solve A by running it on $f(x)$.

2 Cvičení 2010-10-04

2.1 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 jednoprocessorovém 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.

2.2 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:

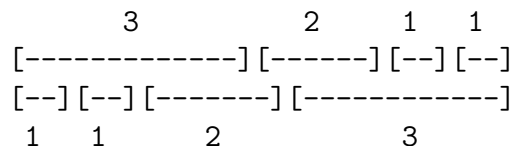
```

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

```

- Iterativní použití kritéria 3 s rozdílem, že koliduje s největším počtem úloh

Protipříklad:

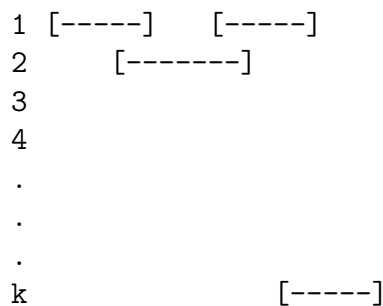


Čísla udávají s kolika úlohami daná úloha koliduje. Je vidět, že opět algoritmus vrátí tři množiny úloh, kdežto evidentně je možné je rozdělit do dvou.

- 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.

Příklad:



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.

3 Matroidy

3.1 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.

3.2 Příklad

- Nechť S je konečná a neprázdná.
- Nechť S_1, \dots, 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 .

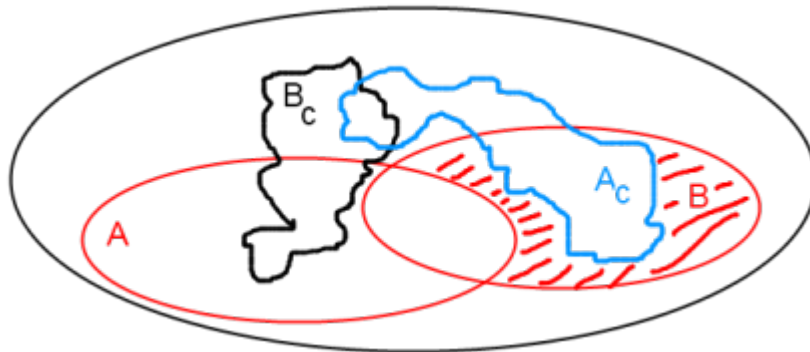
$$\begin{aligned}|B| > |A| &\Rightarrow |\{i \mid |B \cap S_i| = 1\}| > |\{i \mid |A \cap S_i| = 1\}| \\&\Rightarrow \exists i : (|B \cap S_i| = 1) \& (|A \cap S_i| = 0) \\&\Rightarrow \text{Nechť } x \in B \cap S_i \text{ pak } A \cup \{x\} \in I\end{aligned}$$

3.3 Příklad (DÚ)

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

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 $|A \setminus (A \cap B)| < |B \setminus (A \cap B)|$. 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.

□



- Necht' $B_{CC} = B_C \setminus A$

- Necht' $A_{CC} = A_C \setminus B$
- 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 mezi 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.

4 Cvičení 2010-10-18

Příklad 1 Necht' 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: $\text{indegree}(x) = n - 1$ a
- pro výstupní stupeň x platí $\text{outdegree}(x) = 0$,

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:

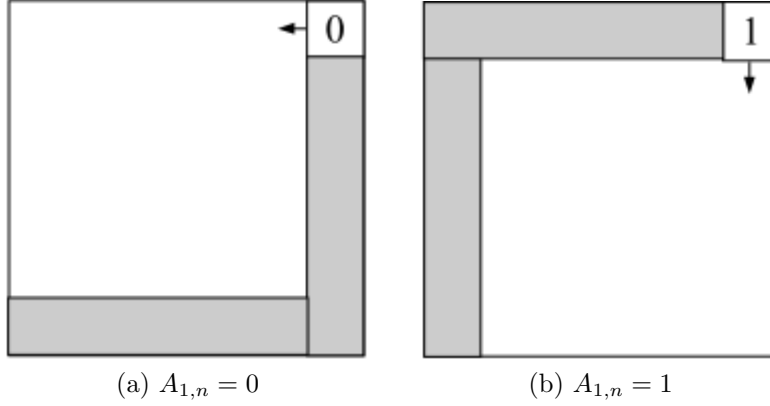
$$\begin{aligned}(i, j) \in E &\Rightarrow A_{i,j} = 1 \\ (i, j) \notin E &\Rightarrow A_{i,j} = 0\end{aligned}$$

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.

Začneme v pravém horním rohu¹ 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?

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



Obrázek 1: Dvě možné výchozí pozice

- 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 $\text{indegree}(j) \leq n - 2$.
- 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 $\text{outdegree}(i) \geq 1$.

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 doleva². 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.

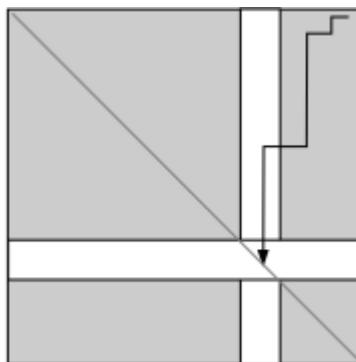
Prvek, na kterém skončí procházka maticí, vidíme na následujícím obrázku.

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 algoritmu:

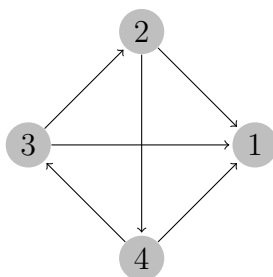
$$T(n) = \Theta((n - 1) + (n - 2) + (n - 1)) = \Theta(n)$$

²Platí, že $D + L = n - 1$.



Obrázek 2: Konec procházky maticí

Příklad: Mějme graf:



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 spustit³ 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 prohledáním 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 $O(n+m)$

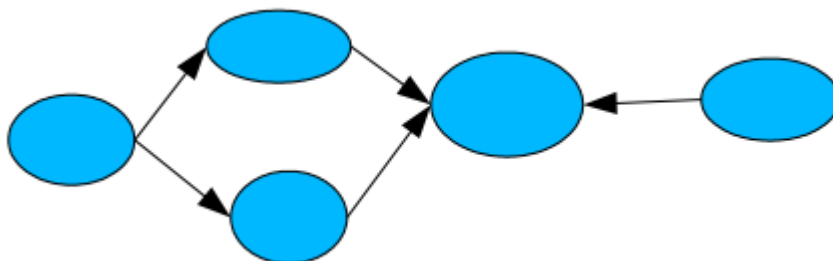
- (a) Nejdříve najdeme v grafu G silně souvislé komponenty S_1, \dots, S_p , což stihneme v čase $O(n+m)$. Algoritmus na hledání silně souvislých komponent:

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
```

👁 **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.

³Časová složitost DFS je $O(n+n)$.



Obrázek 3: Silně souvislé komponenty

- (b) Silně souvislé komponenty bychom mohli uspořádat pomocí topologického třídění⁴. 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 podíváme⁵ 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 .

⁴Místo vrcholů bychom třídili silně souvislé komponenty. Mohli bychom algoritmus použít díky pozorování.

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

5 Cvičení 2010-11-08

Příklad 1 Necht' máme k dispozici „černou skříňku“, která umí řešit SAT (rozhodovací problém splnitelnosti CNF formulí) v polynomiálním čase. Skříňka odpovídá pouze ANO-NE. Zkonstruuje 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. Pust' 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) dosad' $x = 0$ do aktuálního CNF a pust' 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) dosad' $x = 1$ 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 incidentní⁶ 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 $k \in \mathbb{N}$

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 $k + 1, k + 2, \dots, |V|$, naopak to obecně neplatí.

⁶Jestliže $(u, v) \in E$, 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 $0, 1, \dots$ 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.

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, \dots, 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 $\text{TAUT} \in \text{NP}$, pak $\text{NP} \equiv \text{co-NP}$. 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. Na formuli "útočíme po klauzulích". Stačí aby se v jedné klauzuli nevyskytovala nějaká proměnná x_1, \dots, x_n jako komplementární pár⁷ 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:

4. Pozorování: F je nesplnitelná $\Leftrightarrow \neg F$ je nesplnitelná. TODO

6 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.

⁷Tedy ve formě gace a zároveň ve formě negace.

Řešení Dokazujeme, že LOUP je NP-úplný problém.

1. LOUP \in NP

- Certifikát: Množina T .

2. Je LOUP NP-těžký?

Převédeme na problém *součtu podmnožiny*⁸ s tím, že součet je $b := \frac{\sum a_i}{2}$ (dělíme půlky). Tedy:

SP [znak redukce] 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 řešení⁹. Nyní zdefinujeme 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: $a_1, \dots, a_n, a_{n+1}, b'$ má řešení \Leftrightarrow vstupní instance SP: $\{a_1, \dots, a_n\}, b$ má řešení.

Důkaz.

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

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

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

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

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 $I \subseteq S$ (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:

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

I' je tedy řešením instance LOUP.

□

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

Řešení TODO

Příklad 3 Definujme problém 0-1 CP (celočíslné programování) následovně:

Instance: Celočíslná matice A řádu m krát n a celočísl. 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.

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, \dots, x_n . We will have corresponding variables z_1, z_2, \dots, 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 $x_i = \text{true}$ in the formula, and assigning $z_i = 0$ represents setting $x_i = \text{false}$.

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 $x_1 = \text{true}$ or $x_2 = \text{false}$ or $x_3 = \text{true}$ in the corresponding truth assignment. \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) ¹⁰

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 vertices 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 $(G_1 = (V_1; E_1); G_2 = (V_2; E_2); \phi : V_1 \rightarrow V_2)$. The verifying algorithm checks if ϕ is a 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 complete graph on k vertices, and G_2 to be the graph G . Then $(G_1; G_2) \in$ SUBGRAPH ISOMORPHISM if and only if $(G; k) \in$ CLIQUE. \square

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

Poznámka: Proč je toto řešení indukované varianty? Protože, pro všechny vrcholy, které tvoří 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 . \square

7 Cvičení 2010-06-12 (5. cvičení)

Příklad 1 Nechť máme k dispozici „černou skříňku“, která umí řešit rozhodovací verzi problému součtu podmnožiny v polynomiálním čase. Skříňka 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.

¹⁰Lze v nezměněné podobě použít i na variantu 1

Formální zápis úlohy:

Vstup: Čísla $x_1, \dots, x_n, t \in \mathbb{Z}^+$.

Výstup: Množina $S \subseteq \{1, \dots, n\}$ taková, že $\sum_{i \in S} x_i \leq t$ a $\sum_{i \in S} x_i$ je maximální.

Řešení

Definujme čísla: y_0, \dots, 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 $x_1, \dots, x_n, y_0, \dots, y_{l-1}$ 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 nachází v intervalu (t - (2^l - 1) ; t)
else {
// maximum se nachází v intervalu (0 ; t - (2^l - 1))
t = t - y_l-1;
}
l--;
```

Ono totiž. Dostavas-li od BB odpovědi ANO, tak se ti cílový interval stále zmenšuje. Když dostaneš odpověď NE, tak vis, že se maximum nenachází už blízko t . Takže dostaneš určitý interval (záleží na počtu dotazu ANO, než přišel NE), kde to maximum je. Odpověď NE pro konkrétní l ti řekne, že není podstatné, zda v množině $y_l - 1$ je či není. Ať už s ním, nebo bez něj,

součet hodnoty t prostě nedosáhne. Takže podle o nej můžeš snížit t -kovu jít do další iterace algoritmu.

Zkus si to nakreslit. Ten interval se bude stále pulit imho. A jakmile získáš přesnou hodnotu maxima, tak už je to jednoduché. Provedeš n dotazů do BB, čímž zjistíš, zda dané x_i se v tom součtu nachází, nebo ne:

Odeberes x_i z množiny a zeptáš se BB:

ANO — x_i není v součtu

NE — x_i je v součtu.

Jen teď nevím, zda jsou tyto kroky zbytečné, nebo ne. Každopádně to znamená $n + \log t$ dotazů na BB, což je polynomiální vzhledem ke vstupu ($n * \log t + \log t + \log t * \log t$), což bude asi vše $O(n)$.

Příklad: Nechť $t = 18$ a dále

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

1. test: $x_1, \dots, x_4, y_0, \dots, y_3, l = 4$

Odpověď BB je ANO, řešení je v intervalu:

$$[t - (2^l - 1), t] = [18 - 15, 18] = [3, 18]$$

2. test: $x_1, \dots, x_4, y_0, \dots, y_2, l = 3$

Odpověď BB je ANO, řešení je v intervalu:

$$[18 - (2^3 - 1), 18] = [11, 18]$$

3. test: $x_1, \dots, x_4, y_0, y_1, l = 2$

Odpověď BB je ANO, řešení je v intervalu:

$$[18 - (2^2 - 1), 18] = [15, 18]$$

4. test: $x_1, \dots, x_4, y_0, l = 1$

Odpověď BB je ANO, řešení je v intervalu:

$$[18 - (2^1 - 1), 18] = [17, 18]$$

5. test: $x_1, \dots, x_4, l = 0$

Odpověď BB je NE, řešení je v intervalu:

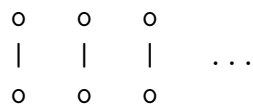
$$[17, 18 - (2^0 - 1)] = [17, 18]$$

Řešením je tedy 17.

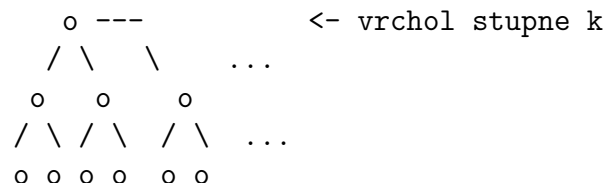
Příklad 2 Popište jak lze pro libovolné dané n zkonstruovat graf na n vrcholech takový, že aproximační algoritmus pro vrcholové pokrytí s poměrovou chybou $r = 2$ (prezentovaný na přednášce) na tomto grafu vrací pokrytí právě dvakrát větší než je velikost optimálního vrcholového pokrytí (tj. ukažte,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

Řešení

- První řešení:



- Budeme-li požadovat navíc souvislý graf:



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

8 Cvičení 2010-01-03 (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 navrhnete polynomiální aproximační algoritmus s poměrovou chybou $r=3$.

Řešení Nejdříve dokážeme, že BTSP je NP-úplný.

9 Zdroje

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