Algoritmická matematika 3

Greedy algoritmy

Petr Osička

DATA ANALYSIS AND MODELING LAB
Univerzita Palackého v Olomouci

Zimní semestr 2013

Základní idea

Obecné schéma greedy algoritmu

1. Pro vstupní instanci I algoritmus provede volbu x a na jejím základě vytvoři *jednu* menší podinstanci I' . (Volba x je v tomto popisu abstraktní pojem, v reálném algoritmu to je reálný objekt, např. hrana grafu, číslo, množina, posloupnost bitů apod.)
2. Algoritmus poté rekurzivně aplikujeme na I'. Řešení, které získáme pro I' pak zkombinujeme s volbou x z předchozího kroku a obdržíme tak řešení pro I.
3. Rekurze končí v okamžiku, kdy je vstupní instance dostatečně malá.

Volba není založena na jejích možných důsledcích pro další běh algoritmu, je založena pouze na její **ceně v momentě volby**. Odtud pochází označení greedy (žravé) algoritmy. Algoritmus bez rozmyslu, žravě, vybírá tu aktuálně nejlepší možnost.

Základní idea

```
procedure Greedy(Input)
    I ← Input
    i ← 0
    while |J| ≥ c do
        x_i ← Greedy-Choice(I)
        I ← Create-Subinstance(I, x_i)
        i ← i + 1
    end while
    x_i ← BasicAlgoritm(I)
    Zkombinuj volby x_j, (j = 0, ..., i) do řešení x
    return x
end procedure
```

Nalezení minimální kostry

Definice

Nechť $G = (V, E)$ je neorientovaný spojitý graf. **Ohodnocení hran** grafu G je zobrazení $c : E \to \mathbb{Q}$ přiřazující hranám grafu jejich racionální hodnotu, $c(e)$ je pak ohodnocení hrany $e \in E$. Dvojici (G, c) říkáme hranově ohodnocený graf.

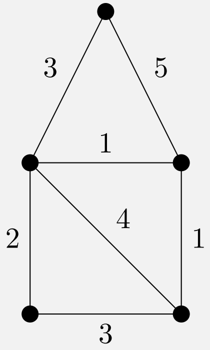
Kostra grafu G je podgraf $G' = (V, E')$ (G' má stejnou množinu uzlů jako G!) takový, že

- G' neobsahuje kružnici,
- G' je spojitý graf. Pro každou dvojici uzlů u, v platí, že mezi nimi existuje cesta

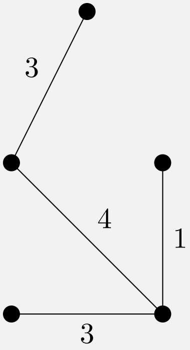
Cena kostry $G' = (V, E')$ v ohodnoceném grafu je součet ohodnocení jejích hran, tj. $\sum_{e \in E'} c(e)$. □

Nalezení minimální kostry

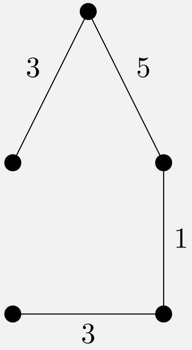
Příklad



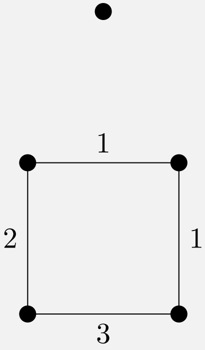
(a) Ohodnocený graf



(b) Kostra s cenou 11



(c) Kostra s cenou 12



(d) Neni kostra

Nalezení minimální kostry

Minimální kostra grafu	
Instance:	Hranově ohodnocený graf (G, c), kde $G = (V, E)$.
Přípustná řešení:	$sol(G, c) = \{(V, E') \mid (V, E')$ je kostra grafu $G\}$
Cena řešení:	$cost((V, E'), G, c) = \sum_{e \in E'} c(e)$
Cíl:	minimum

- existuje několik algoritmů, které najdou optimální řešení v polynomickém čase
- ukážeme si **Kruskalův algoritmus**

Kruskalův algoritmus

Princip

- začneme s $E' = \emptyset$ a v každém kroku do E' jednu hranu přidáme. Při výběru přidané hrany se řídíme jednoduchým pravidlem:

  Vyber hranu s nejmenší cenou, jejíž přidání do E' nevytvoří v (V, E') kružnici.
- Z původního grafu poté tuto hranu a všechny hrany s menší cenou, jejichž výběr by vedl k vytvoření kružnice, odstraníme.
- Iterujeme tak dlouho, dokud nenalezneme kostru.

Při implementaci tohoto algoritmu je potřeba nalézt efektivní způsob hledání hrany s minimální cenou a také ověření existence kružnice.

- Hrany setřídíme
- Použijeme vhodnou datovou strukturu – disjoint set structure

Disjoint set structure

Slouží k uložení kolekce $\mathcal{S} = \{S_1, \ldots, S_n\}$ disjunktních množin.

Každou z množin $S_1, \ldots, S_n$ lze identifikovat pomocí **reprezentanta**, vybraného prvku z dané množiny.

Operace nad strukturou.

- MakeSet(x) přidá do systému novou množinu, jejímž jediným prvkem je x.
- Union(x, y) v systému množin sjednotí množinu, která obsahuje prvek x s množinou obsahující prvek y (původní množiny ze systému odstraní a nahradí je jejich sjednocením).
- FindSet(x) vrátí reprezentanta množiny obsahující x.

Disjoint set structure - Implementace

- Jednotlivé množiny v systému reprezentujeme pomocí kořenových stromů.
- V každém uzlu uchováváme
  - jeden prvek,
  - ukazatel $parent$ na předka ve stromu
  - **rank**, což je horní limit výšky podstromu generovaného daným uzlem.

Poznámky

V následujícím pseudokódu předpokládáme, že argumenty procedur jsou uzly stromu. Je vhodné udržovat všechny uzly odpovídající prvkům z množin, které jsou v systému, i v jiném struktuře s přímým přístupem, např. v poli. Funkce pro operace se strukturou pak pouze mění ukazatele a ranky.

Disjoint set structure - Implementace

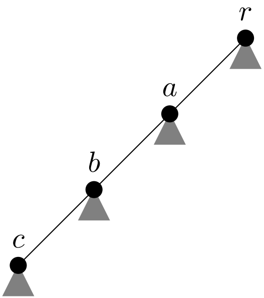
```
procedure MakeSet(x)
    x.parent ← x
    x.rank ← 0
end procedure
```

```
procedure FindSet(x)
    if x ≠ x.parent then
        x.parent ← FindSet(x.parent)
    end if
    return x.parent
end procedure
```

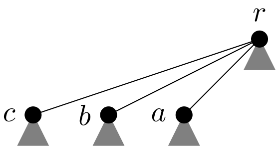
```
procedure Union(x, y)
    r_x ← FindSet(x)
    r_y ← FindSet(y)
    if r_x.rank > r_y.rank then
        r_y.parent ← r_x
    else
        r_x.parent ← r_y
        if r_y.rank = r_x.rank then
            r_y.rank = r_y.rank + 1
        end if
    end if
end procedure
```

Složitost disjoint set structure

- kritická je operace FINDSET, jejíž složitost závisí na výšce stromu.
- heuristiky ve FINDSET a v UNION udržují výšku stromu malou.
- Složitost sekvence m operací se strukturou, z nichž n operací je MAKESET, je $O(m \cdot \alpha(n))$, kde α je inverzní Ackermanova funkce (v praxi téměř vždy menší než 4).



(e) Původní strom



(f) Strom po provedení FINDSET(c)

Kruskalův algoritmus

Disjoint set structure v Kruskalově algoritmu

- Kruskalův algoritmus využívá disjoint set structure pro ukládání množin uzlů grafu.
- Na začátku algoritmu přidáme do systému pro každý vrchol jednoprvkovou množinu.
- Vždy, když v průběhu algoritmu přidáme do řešení novou hranu, sjednotíme množiny, které obsahují koncové vrcholy této hrany.
- V libovolném okamžiku obsahuje každá množina právě vrcholy jedné komponenty grafu tvořeného algoritmem zatím vybranými hranami.
- existenci kružnice testujeme tak, že ověříme jestli jsou s přídávanou hranou incidující uzly ve stejné množině.

procedure KRUSKAL$((G = (V, E), c))$
    Vytvoř prioritní frontu hran Q
    Vytvoř $|V|$ prvkové pole A
    for $i \leftarrow 1$ to $|V| - 1$ **do**
        MAKESET$(A[i])$
    end for
    $E' \leftarrow \emptyset$
    while $|E'| < |V| - 1$ **do**
        Odeber z Q hranu (u, v)
        if FINDSET$(A[u]) \neq$ FINDSET$(A[v])$ **then**
            $E' \leftarrow E' \cup \{(u, v)\}$
            UNION$(A[u], A[v])$
        end if
    end while
    return (V, E')
end procedure

Kruskalův algoritmus

Věta

KRUSKAL *vrací kostru grafu.*

Důkaz.

Množina E' obsahuje $|V| - 1$ hran (řádek 8 algoritmu) a neobsahuje cykly (řádek 10 algoritmu). Tvrzení pak plyne ze souvislosti grafu G. □

Věta

KRUSKAL *vrací minimální kostru.*

Důkaz.

Dokážeme, že po každém přidání hrany do E' existuje minimální kostra $T = (V, B)$ taková, že obsahuje doposud algoritmem nalezené hrany. Důkaz provedeme indukcí přes velikost E'. Označme si jako E'_i i-prvkovou množinu hran, kterou získáme po přidání i-té hrany, kterou si označíme e_i.

Pro $E'_0 = \emptyset$ je situace triviální, stačí vybrat libovolnou minimální kostru.

Předpokládejme, že tvrzení platí pro E'_{i-1} a $T = (V, B)$ je odpovídající minimální kostra. Pokud $e_i \in B$, je tvrzení triviální. Pokud $e_i \notin B$, pak přidáním e_i do B vytvoříme v T kružnici. Pak ale B obsahuje hranu $e_j \notin E'_i$, která leží na této kružnici (jinak by algoritmus nemohl e_i přidat do E'_{i-1}, v E'_i by vznikla kružnice). Potom $(V, B - \{e_j\} \cup \{e_i\})$ tvoří kostru se stejnou cenou jako T.

$\square$

Pokračování důkazu.

Stačí si uvědomit, že po přidaní e_i do B leží obě hrany e_i a e_j na kružnici, a tudíž odstraněním jedné z nich dostaneme kostru. Dále platí, že $c(e_i) \leq c(e_j)$, protože v opačném případě by si algoritmus vybral e_j místo e_i. Skutečně, protože $E'_{i-1} \subseteq B$ tak by přidáním e_j do E'_{i-1} nevnikla kružnice (e_j totiž neleží na kružnici ani v T) a algoritmus tedy e_j nemohl v předchozích iteracích vynechat. Současně T je minimální kostra, takže $c(e_j) \leq c(e_i)$, odtud již dostáváme požadovanou rovnost. □

Složitost

- Vytvoření prioritní fronty má za $O(|E| \log |E|)$.
- Řádek 3 se dá provést s lineární složitostí $O(|V|)$.
- Poté algoritmus provede nejhůře $|V| + 3|E|$ operací nad disjoint sets structure, z nichž $|V|$ operací je MakeSet, dostáváme složitost $O(|E|)$.
- Řádky 9 a 11 mají konstatní složitost.
- Protože u spojitého grafu je $|E| \geq |V| - 1$, můžeme konstatovat, že složitosti dominuje $O(|E| \log |E|)$.

Sestavení Huffmanova kódu

Huffmanův kód je klíčem k efektivnímu uložení řetězců nad určitou abecedou pomocí sekvence bitů (a tedy efektivní uložení tohoto řetezce v počítači).

Definice

Kód nad abecedou Σ je injektivní zobrazení $\gamma : \Sigma \to \{0, 1\}^*$.

Řekneme, že kód γ je **jednoznačný**, pokud existuje jednoznačný způsob, kterým lze libovolné slovo $w = w_1 w_2 \dots w_k \in \Sigma^*$ dekódovat z jeho zakódování $\gamma(w) = \gamma(w_1)\gamma(w_2)\dots\gamma(w_k)$. Tato podmínka je ekvivalentní tomu, že každá dvě různá slova mají různé zakódování.

Kód se nazývá **blokový**, pokud pro každé dva znaky $a, b \in \Sigma$ platí, že $|\gamma(a)| = |\gamma(b)|$.

Sestavení Huffmanova kódu

Příklad

(a) Uvažujme jednoduchou abecedu $\Sigma = \{x, y, z\}$ a kód γ daný $\gamma(x) = 0, \gamma(y) = 1, \gamma(z) = 01$. Je snadno vidět, že kód není blokový. Také není jednoznačný. Uvažme například slovo xyz. Jeho zakódování $\gamma(xyz) = 0101$ lze dekódovat také jako $xyxy$.

(b) ASCII tabulka je příkladem jednoznačného blokového kódu. Každý z 256 možný znaků, které se v tabulce nacházejí má přidělen unikátní sekvenci 8 bitů. Všimněme si, že blokový kód je vždy jednoznačný.

Sestavení Huffmanova kódu

Uvážíme $\Sigma = \{a, b, c, d\}$, pak lze jednoduše vytvořit blokový kód s délkou zakódování jednoho slova 2 bity.

Uvažme řetezec w nad Σ, který obsahuje 100 000 znaků, a znak a v něm má 10 000 výskytů, b má 50 000 výskytů, c má 35 00 výskytů a d má 5 000 výskytů. Pokud zakódujeme w pomocí zmíněného blokového kódu, dostaneme 200000 bitů.

Uvažme kódování, kdy a zakódujeme pomocí 3 bitů, b pomocí 1 bitu, c pomocí 2 bitů, d pomocí 3 bitů. Zakódování w pomocí takového kódování má pak

$$3 \cdot 10000 + 1 \cdot 50000 + 2 \cdot 35000 + 3 \cdot 5000 = 165000 \text{ bitů.}$$

Ušetřili jsme tedy 35000 bitů, což je 17.5 procenta z původní velikosti souboru.

Zohlednění frekvence výskytů znaků v řetezci vede k efektivnějšímu kódu!

Definice

Nechť Σ je abeceda. Kód γ je **prefixový**, pokud pro všechna $x, y \in \Sigma$ platí, že $\gamma(x)$ není prefixem $\gamma(y)$.

Věta

Každý prefixový kód je jednoznačný.

Důkaz.

Stačí ukázat, že existuje procedura pro jednoznačné dekódování. Slovo $w = w_1 w_2 \dots w_n$ dekódujeme ze sekvence $\gamma(w) = b_1 \dots b_m$ následujícím postupem.

1. čteme sekvenci zleva doprava
2. když přečteme sekvenci $b_1 \dots b_j$ takovou, že $\gamma(w_1) = b_1 \dots b_j$ pro dekódujeme w_1
3. smažeme $b_1 \dots b_j$ ze sekvence a pokračujeme bodem 1. Iterujeme dokud není sekvence prázdná.

Díky tomu, že je γ prefixový kód, nemůžeme v bodě 1 dekódovat jiný znak než w_1 (a v dalších iteracích $w_2, w_3, \dots$). $\square$

Příklad

Uvažujme prefixový kód γ daný následující tabulkou.

Symbol	γ
a	11
b	01
c	001
d	10
e	000

Řetez $cecab$ pak kódujeme pomocí 0010000011101. Dekódujeme jako

Krok	$\gamma(cebab)$	dekódovaný znak
1	$\underline{001}0000011101$	c
2	$\underline{000}0011101$	e
3	$\underline{001}1101$	c
4	$\underline{11}01$	a
5	$\underline{01}$	b

Efektivita kódu

Pro $x \in \Sigma$ je **frekvence** f_x znaku x v textu $w \in \Sigma^*$ o n znacích je podíl

$$f_x = \frac{\text{počet výskytů } x}{n}.$$

Efektivitu kódu měříme délkou zakódování vstupního řetězce. Protože

$$|\gamma(w)| = \sum_{x \in S} n \cdot f_x \cdot |\gamma(x)| = n \sum_{x \in S} f_x \cdot |\gamma(x)|$$

můžeme vypustit závislost na délce $|w| = n$ a měřit efektivitu γ pomocí **průměrné délky zakódování jednoho znaku**

$$ABL(\gamma) = \sum_{x \in S} f_x \cdot |\gamma(x)|.$$

Efektivita kódu

Příklad

(a) Uvažme prefixový kód daný následující tabulkou

x	$\gamma(x)$	f_x
a	11	.32
b	01	.25
c	001	.20
d	10	.18
e	000	.05

Průměrná dékla slova tohoto kódu je

$$ABL(\gamma) = 0.32 \cdot 2 + 0.25 \cdot 2 + 0.20 \cdot 3 + 0.18 \cdot 2 + 0.05 \cdot 3 = 2.25$$

Oproti blokovému kódu, který by měl délku ABL rovno 3 (proč?), jsme ušetřili 0.75 bitu.

Hufmanův kód

Definice

Nechť Σ je abeceda znaků vyskytujích se v textu s frekvencemi f_x pro $x \in \Sigma$. Řekneme, že prefixový kód γ kódující Σ je **optimální**, jesliže pro všechny ostatní prefixové kódy γ' platí, že $ABL(\gamma) \leq ABL(\gamma')$. Optimální kód také nazýváme Huffmanův kód.

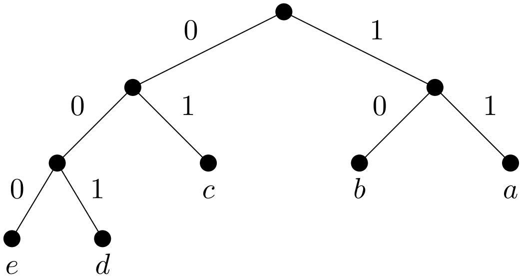
Nalezení Huffmanova kódu

Instance:	abeceda Σ, frekvence výskytu jednotlivých znaků f_x pro $x \in \Sigma$
Přípustná řešení:	$\{\gamma \mid \gamma$ je prefixový kód pro $\Sigma\}$
Cena řešení:	$cost(\gamma, \Sigma, \{f_x \mid x \in \Sigma\}) = ABL(\gamma)$
Cíl:	minimum

Stromová reprezentace prefixového kódu

Pro prefixový kód γ nad abecedou Σ označíme takový strom T_γ.

x	$\gamma(x)$
a	11
b	10
c	01
d	001
e	000



Délka kódového slova $\gamma(x)$ odpovídá v T_γ hloubce listu x (ozn. $depth_T(x)$). Průměrnou délku kódového slova můžeme vyjádřit

$$ABL(T) = \sum_{x \in \Sigma} f_x \cdot depth_T(x)$$

Stromová reprezentace prefixového kódu

Věta

Pro každý optimální prefixový kód γ platí, že každý nelistový uzel v T_γ má stupeň 2.

Důkaz.

Dokážeme sporem. Předpokládejme, že ve stromě T_γ existuje uzel v s jedním potomkem u. Pokud uzel v ze stromu smažeme a nahradíme jej uzlem u, obdržíme strom s menší průměrnou hloubkou (všechny listy v podstromu generovaném uzlem u budou mít o 1 menší hloubku). Tento strom odpovídá prefixového kódu pro stejnou abecedu jako T_γ, protože jsme neodstranili žádný list. To je ale spor s tím, že γ je optimální kód. $\square$

Věta

Nechť γ je optimální prefixový kód. Pak pro každé dva listy y, z ve stromu T_γ platí, že pokud $depth_{T_\gamma}(z) > depth_{T_\gamma}(y)$, pak $f_y \geq f_z$.

Důkaz.

Sporem Pokud $f_y < f_z$, pak prohozením uzlů z a y získáme strom s menší průměrnou hloubkou, což je spor. Skutečně: Podíváme-li se na členy sumy $\sum_{x\in\Sigma} f_x \cdot depth_{T_\gamma}(x)$ pro $x \in \{y, z\}$, pak zjistíme, že

- násobek f_y vzroste z $depth_{T_\gamma}(y)$ na $depth_{T_\gamma}(z)$
- násobek f_z klesne z $depth_{T_\gamma}(z)$ na $depth_{T_\gamma}(y)$

Změna je tedy (rozdíl sumy před a po výměně pro x, y):

$$(f_y \cdot depth_{T_\gamma}(y) - f_y \cdot depth_{T_\gamma}(z)) + (f_z \cdot depth_{T_\gamma}(z) - f_z \cdot depth_{T_\gamma}(y)) =$$
$$(depth_{T_\gamma}(y) - depth_{T_\gamma}(z))(f_y - f_z)$$

Poslední výraz je vždy kladný, proto má nový strom menší průměrnou hloubku. □

Stromová reprezentace prefixového kódu

Věta

Existuje optimální prefixový kód γ takový, že listy x, y v T_γ takové, že f_x a f_y jsou dvě nejmenší frekvence, jsou

(a) v maximální hloubce

(b) sourozenci

Důkaz.

(a) Plyne z předchozí věty.
(b) Prohazováním listů, které jsou ve stejné hloubce se nezmění průměrná hloubka listů. Protože v T_γ mají všechny nelistové uzly stupeň 2, musí existovat v maximální hloubce dva listy, které jsou sourozenci. Tyto listy pak můžeme prohodit s x a y. $\square$

Huffmanův kód

Algoritmus HUFFMAN **pro konstrukci** T_γ

- udržuje si množinu stromů,
- kořen každého má přiřazeno číslo – sumu frekvencí znaků abecedy, které jsou listy stromu.
- Na začátku algoritmu vytvoříme pro každý znak z abecedy jednoprvkový strom, frekvence nastavíme na frekvence výskytů odpovídajích znaků.
- Poté algoritmus greedy strategií sestavuje výsledný strom:
  1. Vybere dva stromy x, y s nejnižšími frekvencemi kořenů f_x a f_y a spojí je do nového stromu tak, že vytvoří nový kořen w, nastaví jeho frekvenci na $f_w = f_x + f_y$. Kořeny stromů x a y se pak stanou potomky w.
  2. Opakuje předchozí krok, dokud nespojí všechny stromy do jednoho.

Optimalita

Věta

Nechť $S = \{x_1, \dots, x_n\}$ *je abeceda znaků s frekvencemi* $f_{x_1} \dots f_{x_n}$ *a* $S' = S - \{x_i, x_j\} \cup \{w\}$*, kde* x_i, x_j *jsou znaky s nejmenšími frekvencemi a* w *je nový znak s frekvencí* $f_w = f_{x_i} + f_{x_j}$*. Nechť* T' *je strom optimálního kódu pro* S'*. Pak pro strom* T*, který dostaneme z* T' *tak, že nahradíme* w *vnitřním uzlem s potomky* x_i*,* x_j *platí:*

(a) $ABL(T') = ABL(T) - f_w$

(b) T *je strom optimálního kódu pro abecedu* S*.*

Věta

HUFMANN *vrací optimální kód.*

Důkaz.

Stačí si všimnout, že jeden krok algoritmu odpovídá konstrukci z předchozí věty. □

Důkaz.

(a) Hloubky všech listů mimo x_i a x_j sou stejné v T i T'. Hloubka listů x_i a x_j v T je o 1 větší než hloubka w v T'. Odtud máme, že

$$
\begin{aligned}
ABL(T) &= \sum_{x \in S} f_x \cdot depth_T(x) \\
&= f_{x_i} \cdot depth_T(x_i) + f_{x_j} \cdot depth_T(x_j) + \sum_{x \neq x_i, x_j} f_x \cdot depth_{T'}(x) \\
&= (f_{x_i} + f_{x_j})(1 + depth_{T'}(w)) + \sum_{x \neq x_i, x_j} f_x \cdot depth_{T'}(x) \\
&= f_w + f_w \cdot depth_{T'}(w) + \sum_{x \neq x_i, x_j} f_x \cdot depth_{T'}(x) \\
&= f_w + \sum_{x \in S'} f_x \cdot depth_{T'}(x) = f_w + ABL(T')
\end{aligned}
$$

$\square$

Důkaz.

(b) Sporem. Předpokládejme, že kód odpovídající T není optimální. Pak existuje optimální kód se stromem Z tak, že $ABL(Z) < ABL(T)$. Podle věty 6 můžeme bez obav předpokládat, že x_i a x_j jsou v Z sourozenci. Označme jako Z' strom, který získáme ze Z náhradou podstromu generovaným rodičem uzlů x_i a x_j pomocí nového uzlu s frekvencí $f_w = f_{x_i} + f_{x_j}$. Pak podle (a) máme:

$$ABL(Z') = ABL(Z) - f_w < ABL(T) - f_w = ABL(T').$$

To je ale spor s tím, že T' je optimální. □