Algoritmy a datové struktury I, přednáška prof. RNDr. Luďka Kučery, DrSc.

Poznámky sepsal Robert Husák

Letní semestr 2009/2010

Většina úloh je dostupná v Algovizi[2], proto v poznámkách uvádím přímo odkazy na jednotlivé scény. Někdy může být scéna nedostupná, ale naopak text v manuálu[3] užitečný k pochopení, proto je u každé scény napsáno i číslo stránky v manuálu.

Část I
Datové struktury

1 Slovník

Objekty z množiny M jsou zde ukládány podle klíče. Operace:

1. Vytvoření "škatulky" (inicializace): $M := \emptyset$
2. `insert` = vložení objektu do množiny: $M := M \cup \{v\}$ ($v \in M \rightarrow$ nic neudělá, nebo ohlásí chybu; někdy musíme ošetřit, aby se nepřidával)
3. `delete` = vynechání: $M := M - \{v\}$ ($v \notin M \rightarrow$ může spadnout, nebo nic neudělat)
4. `search`: $v \in M$?
5. `min`, `max` - používá se, pokud jsou prvky uspořádané (`+ insert` $\rightarrow$ setřídění podle velikosti)

Reprezentace v poli: `delete` $\rightarrow$ vynechat + posunout zbytek; `insert` $\rightarrow$ zkopírovat do nového pole o délce $n + 1$. Všechny operace - procházení celého pole $--> $ trvá dlouho.

2 Binární vyhledávací strom (BST)

- Jedná se o slovník - implemetuje totiž slovníkové operace
- otec
  - levý syn
  - pravý syn
- list - vrchol, co nemá syna
- pro libovolný vrchol platí: prvky vlevo (tedy prvky levého podstromu) musí být $<$ než vrchol a prvky vpravo

  - $\Rightarrow$ zleva doprava tvoří rostoucí posloupnost
  - snazší `hledání` - celý strom procházím od kořene a v každém vrcholu jedu do L nebo P syna
- `přidávání` - postupuju jako při hledání, když dojdu až do listu, přidám nový vrchol
- `vynechávání` - vyhledám a dále:
  1. pokud měl jenom jednoho syna, vynechám ho a syna napojím na jeho otce (děd - vnuk)
  2. pokud měl dva syny, pouze z něj odstraníme klíč, potom nalezneme maximum v levém podstromě, jeho klíč přesuneme do prázdného vrcholu a zbylý prázdný vrchol odebereme (měl max. 1 syna)
- doba prohledávání je úměrná hloubce stromu, o ní platí:
  - průměrná hloubka $\approx 1,5 \log_2 n$, kde n je počet vrcholů (tedy i pro velká n dostatečně malá)
  - záleží na vhodném zvolení hodnoty kořene (nejlépe medián) a následných vrcholů
  - obecně je však $\mathcal{O}(n)$ jako u spojového seznamu (např., když do BST postupně přidáme celou uspořádanou posloupnost)

Prostředkem pro vyvažování stromu je **rotace hrany** - tato úprava zachovává pořadí zleva doprava, ale mění vertikální podobu stromu. Je to vratná operace - stačí zarotovat stejnou hranu znovu.

Scéna (Str. 37)**.** Rotace - Doporučuji si rotace v této scéně vyzkoušet.

Konec 1. přednášky

2.1 AVL stromy

(Podle jmen autorů Adelson-Velského a Landise.)

- podmínka: výška levého podstromu se od výšky pravého podstromu liší nejvýše o 1
- každý vrchol v sobě nese údaj o vyváženosti (tedy údaj, zda je hlubší jeho levý nebo pravý podstrom, případně jestli jsou stejně hluboké), v Algovizi je to reprezentováno "vahadýlkem"
- má rozumnou hloubku:
  - po bližším zkoumání zjistíme, že počet vrcholů n potřebný k sestavení AVL stromu hloubky h je $\approx F_n$
  - Fibonacciho čísla rostou exponenciálně, proto naopak $h \approx \log n$
- lze jej získat rotací z nevyváženého stromu

Scéna (Str. 39)**.** Rotace v AVL stromu - Můžete si pomocí rotací zkusit vyrobit AVL strom z nevyváženého stromu.

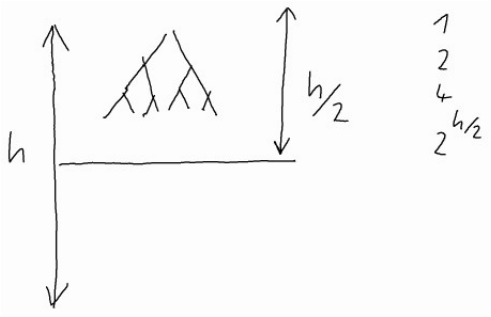
Scéna (Str. 39)**.** Přidávání do AVL stromu - Dělí se na 3 případy.

Scéna (Str. 42)**.** Vynechávání v AVL stromu

2.2 RB stromy

(Red-Black)

- podmínky:
  - kořen je vždy černý
  - otec a syn nemůžou být oba červení
  - jakákoliv cesta z kořene do listu nebo do vrcholu s 1 synem musí obsahovat stejný počet černých vrcholů
- Délka nejdelší cesty tedy nemůže být větší než dvojnásobek délky cesty nejkratší.
- Hloubka:



Obrázek 1: Hloubka RB-stromu

$$N \geq 1 + 2 + .. + 2^{h/2} > 2^{h/2}$$

$$2 \log_2 N \geq h$$

- Pravidlo pro rotace: nelze rotovat hranu, kde je otec červený a syn černý, jiné případy rotovat můžeme.

Konec 2. přednášky

`insert`:

1. Najdeme, kam ho umístit. Přidáme jako červený (pokud to není kořen), pokud splňuje strom podmínky (tj. otec je černý), jsme hotovi.
2. Jinak napravujeme směrem nahoru:
   (a) Pokud je strýc červený (nebo neexistuje), začerníme otce a strýce, začerveníme dědečka.
   (b) V opačném případě:
      i. Pokud je děda ve stejném směru jako otec: provedeme rotaci hrany otce a dědy. Pak můžeme postup ukončit, všechno jsme totiž napravili.
      ii. Pokud ne, je předtím ještě potřeba zarotovat hranu s otcem.

Scéna (Str. 45)**.** Červeno-černé vkládání - Dělí se na 3 případy.

`delete`:

1. Vynechávám vrchol s dvěma syny: vynechám ve vrcholu hodnotu, najdu maximum v levém podstromě, jeho hodnotu přenesu do vynechaného a vynechám ho.
2. Vynechávám vrchol, který má pouze jednoho syna (to musí být červený list a vrchol musí být černý): Přeneseme hodnotu syna do vrcholu a syna vymažeme.
3. Vynechávám červený list: Prostě jej vymažeme (nic se neporuší).
4. Vynechávám černý kořen: Prostě jej vymažeme (nic se neporuší).
5. Vynechávám černý list, kde je otec černý a bratr červený (bratr musí být červený a mít černé syny): Provedeme rotaci hrany otec-bratr. Potom otočený vrchol obarvíme na černo, syny na červeno a tím pádem pouze smažeme červený list.
6. Vynechávám černý list, kde je otec černý a bratr černý a bratr má syna blíže k tomu listu (bratr může mít za syny pouze červené listy): Přepíšeme hodnotu listu hodnotou otce a hodnotu otce hodnotou bratrova syna. Bratrova syna vymažeme.
7. Stejné jako předcházející, ale bratr má syna na opačnou stranu: Vymažeme z listu hodnotu a popřesouváme postupně hodnoty nejbližších vyšších čísel a nakonec vymažeme červený list.
8. Vynechávám černý list, který má červeného otce (bratr je tedy černý) a bratr je list: Prohodím barvu otce a synů (tedy "zvednu nahoru"), vrchol můžu vymazat.
9. Vynechávám černý list, který má černého otce - kořen a bratra černý list: Zvednu nahoru a vymažu.
10. Stejné jako předcházející, ale otec není kořen: Odbarvíme ho i bratra → otec bude dvojnásobně černý. Červený list vynecháme. Musíme však ošetřit dvojnásobný černý list: Pokud má červeného bratra: Provedu rotaci hrany otec-bratr a černou barvu do otočeného červeného bratra přenesu z jeho synů.
11. Stejné jako předcházející, ale strýc je černý a děda je červený (bratranci musí být černí): Ošetříme dvojn.: zvedneme nahoru do červeného dědy.
12. Stejné jako předcházející, ale i děda je černý: Dvojn.: zvedneme nahoru a dále ošetřujeme dvojn. dle situace. Pokud dojdeme do kořene, můžeme dvojitou černou nahradit za černou.
13. Stejné, ale vzdálenější bratranec je červený: "Přesunu černou barvu" ze strýce dolů. Provedeme rotaci děda-strýc a barvu zvedneme nahoru. Pokud byl děda červený, použijeme stejný postup a samo se to spraví.
14. Stejné, ale bližší bratranec je červený: Dvojn.: Provedu rotaci hrany červený bratranec-strýc → předchozí situace. Platí i pro oba bratrance červené, i pro červeného dědu, i pro kombinaci obojího.

Scéna (Str. 48)**.** Červeno-černé vynechávání - Dělí se na 14 případů, ty se však na sebe navzájem odvolávají.

Konec 3. přednášky

3 Hloubka průměrného binárního stromu

Viz PDFko ke stažení [4] na stránkách pana profesora Kučery [1].

Konec 4. přednášky

Část II
Třídící algoritmy

4 Mergesort

5 Heapsort

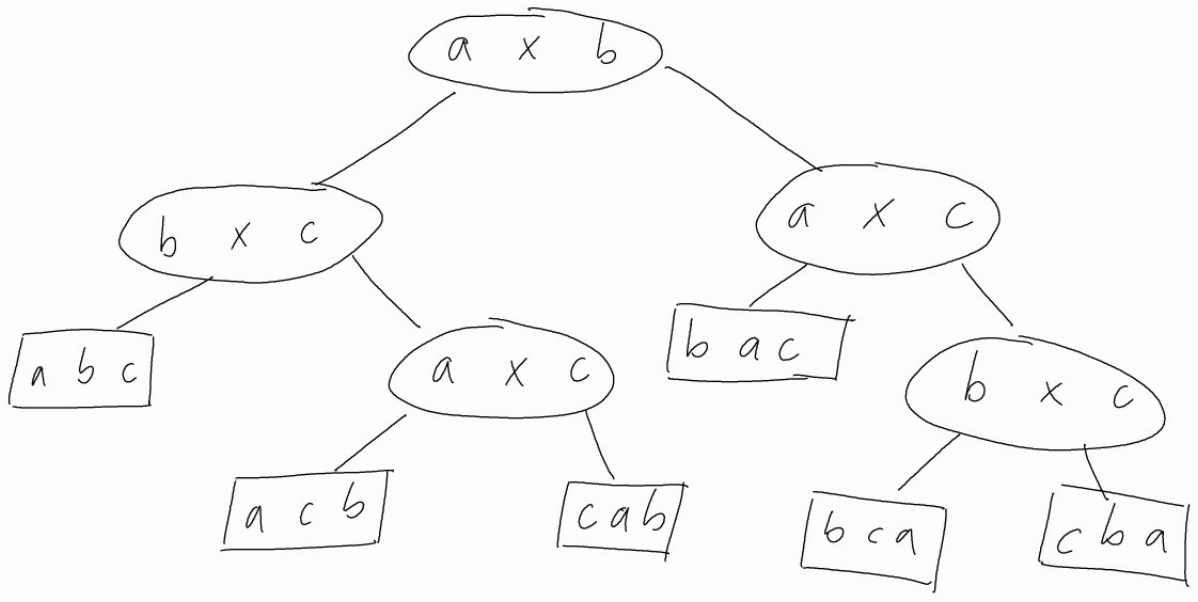
6 Quicksort

Věta 6.1 (Průměrná složitost quicksortu)**.** *Průměrná časová složitost quicksortu je* $\Theta(n \log n)$.

Důkaz. ...

7 Dolní odhady porovnávacího třídění

Pozorování 7.1. Každý deterministický třídící algoritmus lze jednoznačně modelovat binárním rozhodovacím stromem, viz obr. 2. Levá větev vždy odpovídá výsledku "¡" a pravá větev "¿" (BÚNO vstupy různé).



Obrázek 2: Rozhodovací strom pro insertion-sort a $n = 3$ (označme vstup a, b, c)

Pozorování 7.2. Rozhodovací strom modeluje korektní třídící algoritmus $\Rightarrow$ musí obsahovat listy se všemi $n!$ možnými pořadími (permutacemi) vstupní posloupnosti. Počet porovnání v nejhorším případě = nejdelší větev od kořene k listu = výška stromu.

Věta 7.3 (Nejhorší případ)**.** *Binární strom s* $n!$ *listy má výšku* $\Omega(n \log n)$

Důkaz. ... (Část lze dokázat přes Stirlingovu formuli.)

Poznámka 7.4. Chybí mi zápisky z 5. přednášky: "Třídění - časová složitost: mergesort, heapsort, průmerná složitost quicksortu, dolní odhad pro porovnávací třídicí algoritmy (nejhorší prípad)" Nejsem si jistý, co všechno z toho bude chtít dokázat, možná bude chtít přímo "Analýzu složitosti rekurentních algoritmů", viz zápisky Jana Procházky[10]. Výpisky výše jsou z prezentace pana docenta Čepka[9].)

Konec 5. přednášky

Věta 7.5 (Průměrný případ)**.** *Je li A porovnávací algoritmus, pak průměr přes všechny permutace* π *množiny* $\{1, ..., n\}$ *z počtu porovnání, které A potřebuje pro setřídění posloupnosti* $\pi(1), ..., \pi(n)$ *je alespoň* $\log_2(n!)$*.*

Důkaz. Viz PDFko ke stažení [5] na stránkách pana profesora Kučery [1].

8 Radix sort

Chybí znění algoritmu.

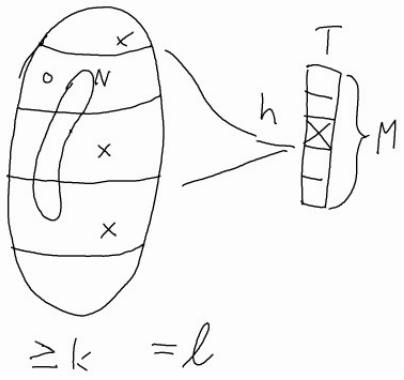
Konec 6. přednášky

Část III
Hashování

Chybí 7. přednáška.

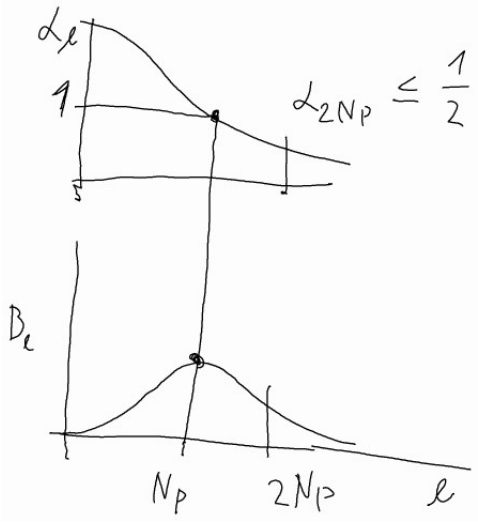
Konec 7. přednášky

Definice 8.1 (Binomické rozložení)**.** Všechny možné podmnožiny úspěšných pokusů ze všech pokusů: $\binom{N}{l} p^l (1-p)^{N-l}$



Obrázek 3: Princip hashování

Důkaz (Dokončení důkazu tvrzení o průměrném případě v univerzálním hashování)**.** (**Bude ho prý chtít pouze po těch, kteří chtějí jedničku**) Celý důkaz je zapsán v PDFku[6] na stránkách[1].



Obrázek 4:

$$B_l = \mathcal{M} \sum_{l=K}^{N} \binom{N}{l} p^l (1-p)^{N-l}$$

p - poměr velikosti U a částí U, která se zhashuje na stejný klíč.

$$N \approx M$$

$$p \approx \frac{1}{M}$$

Tvrdíme, že $\exists$ konstanta C taková, že když $k = C \log N$, pak je pravděpodobnost malá.

$$\alpha_l = \frac{B_{l+1}}{B_l} = \frac{N-l}{l+1} p \frac{1}{1-p} = \frac{N-l}{l+1} \frac{p}{1-p} \overset{l=Np}{=} \frac{N-Np}{Np+1} \frac{p}{1-p} = \frac{(1-p)}{p+1/N} \frac{p}{1-p}$$

Pro $N >> p$ je $\alpha_l = \frac{p}{p} = 1$. V rozmezí $l = Np - 1$ až Np α_l projde jedničkou. Největší pravděpodobnost, že z N pokusů s pravděpodobností P uspěji l-krát, je pro $l = Np$. Pro $M = 2Np$ je $\alpha_l \leq \frac{1}{2}$, čili $B_l \geq \frac{1}{2} B_{l+1}$.

$$k \geq N_p$$

$$\sum_{l=k}^{N} B_l \leq B_k + \frac{1}{2} B_k + \frac{1}{4} B_k + ... \leq 2B_k$$

$$l = 2Np + j$$

$$B_l \leq \frac{1}{2} B_{l-1} \leq \frac{1}{4} B_{l-2} \leq ... \leq \frac{1}{2^j} B_{np} \leq \frac{1}{2^j} = \frac{1}{N^C}$$

$$l = 2Np + C \log_2 N \Rightarrow B_l \leq \frac{1}{2^{C \log_2 N}} = \frac{1}{N^C}$$

Část IV
Grafové algoritmy

9 Nalezení minimální kostry

Kroky si rozdělíme na sudé a liché, v každém lichém kroku přidáváme hranu (říkáme, že ji z původních hran přidáváme do tzv. **výpočetního lesa**):

1. V každém sudém kroku si vyberu komponentu a rozšířím okolí okolo všech jejích bodů.
2. Nejbližší bod s ní v lichém kroku spojím.

Jednotlivé algoritmy se liší pouze pravidly, podle kterých se v sudém kroku vybírá komponenta.

Scéna (Str. 158)**.** Obecný algoritmus
Pro zjišťování vzdálenosti se používají různé metriky:

1. Euklidovská: $\sqrt{(x_1 - x_2) + (y_1 - y_2)}$
2. $\mathcal{L}_1$: $|x_1 - x_2| + |y_1 - y_2|$
3. $\mathcal{L}_{\max}$: $\max |x_1 - x_2|, |y_1 - y_2|$

Scéna (Str. 160)**.** Metrika $\mathcal{L}_1$ a $\mathcal{L}_{\max}$

Algoritmus 9.1 (Jarník - Prim)**.** *Vycházíme z jedné komponenty a tu rozšiřujeme. Výhodná datová struktura pro uchovávání hran vedoucích z této komponenty je například halda.*

Scéna (Str. 161)**.** Jarníkův - Primův algoritmus

Algoritmus 9.2 (Kruskal)**.** *V sudém kroku rozšiřuji okolí okolo všech komponent, v lichém kroku spojím dvě "nejlevnější" komponenty (tedy ty, jejichž okolí se protnou jako první). Aby bylo patrné, že se jedná o implementaci obecného schématu, můžeme algoritmus pojmout tak, že právě jednu z těchto dvou komponent v sudém kroku vybereme a proto ji potom v lichém kroku musíme spojit právě s tou druhou.*
Jiný popis:

1. *Setřídíme si povolené hrany podle velikosti*
2. *Postupně přidáváme hrany. Kdyby měla spojovat již propojené komponenty, vyřadíme ji.*
3. *Když je vše propojené, zbylé možné hrany se zahodí.*

Scéna (Str. 161)**.** Kruskalův algoritmus - V následující scéně je ukázán ještě druhý způsob, jak jej počítat.

Konec 8. přednášky

Důkaz (Správnost obecného schématu)**.** ...

Scéna (Str. 163)**.** Správnost - Celý důkaz je vysvětlen v knize a teď už funguje i v Algovizi.

10 Faktorová množina

Slouží k rozdělení pevné množiny M do navzájem disjunktních tříd. Operace:

- `inicializace`: Nastaví ji tak, že popisuje rozdělení M do jednoprvkových tříd.
- `určení třídy`: Jednoznačně určit, v jaké třídě se $x \in M$ nachází.
- `sloučení tříd`: Dvě dané různé třídy rozkladu se sloučí v jednu. Lze jej provést pouze $(n-1)$-krát, kde n je počet vrcholů.

10.1 Rozklad obarvením

- Každému prvku je přiřazeno číslo (barva), které určuje, do které třídy patří.
- Při slučování tříd se vrcholy z jedné třídy přebarví na barvu vrcholů druhé třídy.
  - Vylepšení: přebarvovat vždy třídu menší velikosti. Potom provedeme celkem maximálně $0,5n \log_2 n$ přebarvení.

Konec 9. přednášky

10.2 Rozklad popsaný stromem

- Spojujeme body do komponent - tvoříme strom.
- Tento strom je jednosměrný a pointery v něm ukazují "zespoda nahoru", tedy sbíhají se k "reprezentantovi".
- Zjišťování, zda jsou dva dané body ve stejné komponentě: od obou bodů dojdeme k reprezentantovi a ty porovnáme. V základní podobě nás tato operace může stát v průměru $\log_2 n$ operací (narozdíl od sloučení, které probíhá v čase konstantním).
  - vylepšení: při procházení k reprezentantovi přepojíme všechny průchozí body rovnou k němu $\rightarrow$ časová složitost:
    $$\mathcal{O}(n \log^* n)$$
    Kde $\log^* n$ je funkce inverzní k věžové funkci.

11 Prohledávání grafu

Rozdělíme si vrcholy na:

- nedosažené
- dosažené
- probrané

a hrany na:

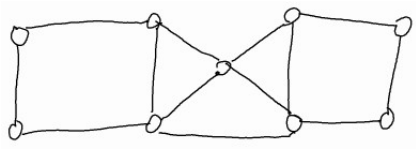
- nepoužité
- použité

procházíme vrcholy s přilehlými hranami, vrcholy, které mají použité všechny hrany, označíme jako probrané.

Dosažené vrcholy jsou uloženy:

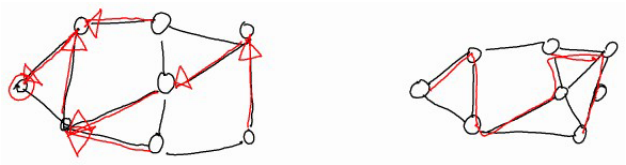
1. v FIFO frontě (tj. fronta) → "prohledávání do šířky" - procházíme graf po vrstvách
2. v LIFO frontě (tj. zásobníku) → "prohledávání do hloubky" - prohledáváme graf jako bludiště
3. v libovolné jiné datové struktuře → prohledáváme graf libovolně

12 Zjišťování stupně spojitosti grafu



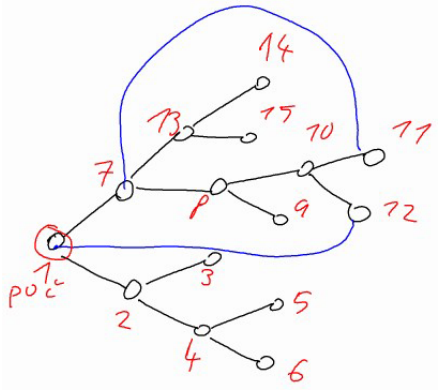
Obrázek 5: 2-souvislý graf

- Sestavíme si "strom prohlevávání", zbylé hrany jsou příčky - při prohledávání do hloubky nám vyjdou příčky tak, že nikdy nespojují body v různých větvích. Tedy libovolný vrchol je artikulace, pokud z něj nevede žádná příčka směrem ke kořenu.



Obrázek 6: Strom prohledávání

- Očíslujeme si vrcholy podle pořadí, v jakém jsme je prohledávali. Navíc zavedeme fci $L(w) := \min\{\text{pořadové číslo } w, \text{pořadové číslo z } |(w,z) \text{ je hrana}\}$ a při prvním průchodu do ní vložíme první odhad: pořadové číslo.



Obrázek 7: Strom prohledávání s očíslováním vrcholů

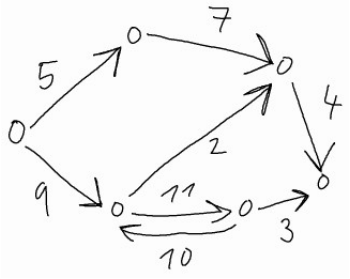
- Při návratu do vrcholu w z podstromu tuto hodnotu ještě "doladíme".
- Z hodnot očíslování a fce L lze zjistit, které body jsou artikulace.

Scéna (Str. 129)**.** Hledání artikulací - V appletech sice nefunguje, ale v knížce je zapsána docela podrobně.

Scéna (Str. 131)**.** Výpočet LOW PT - To samé platí i pro tuto.

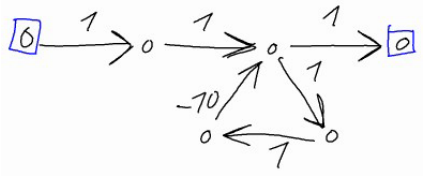
Konec 10. přednášky

13 Hledání nejkratší cesty



Obrázek 8: Orientovaný graf

Poznámka 13.1. Někdy je potřeba hledat nejdelší cestu. V takovém případě přiřadíme ohodnocení hran opačné znaménko, než mají a hledáme cestu nejkratší.



Obrázek 9: Problém záporných hodnot v grafu

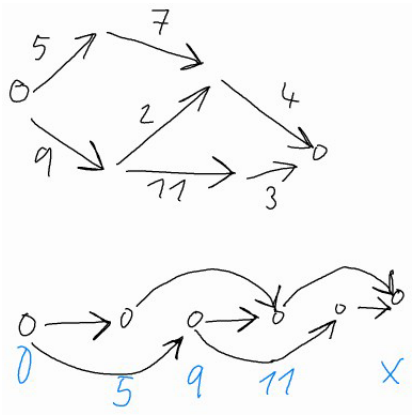
Problémy:

- Může se stát, že nejkratší cesta neexistuje (viz 9). Pak musíme např. zakázat opakování v cyklech apod.
- Problém hledání nejkratší cesty v obecném záporně ohodnoceném grafu je NP-úplný.

Proto se často zadání omezuje:

1. Graf pouze acyklický
2. Neexistují záporně ohodnocené hrany (pro takové grafy známe velmi efektivní algoritmy)
3. Neexistuje záporný cyklus

Definice 13.2. Topologicky uspořádaný graf je acyklický graf uspořádaný tak, že všechny hrany vedou zleva doprava.



Obrázek 10: Topologické uspořádání grafu

Obrázek 11: Nekonečný acyklický graf

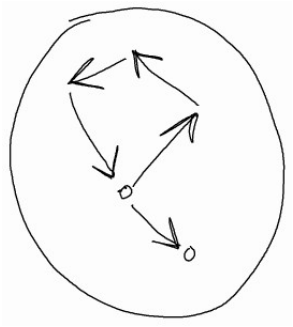
Algoritmus 13.3 (Topologické uspořádání grafu)**.** *Graf musí být konečný (viz obr. 11). Postupným obarvováním vrcholů na zeleno dle algoritmu níže získáme jejich topologické uspořádání - viz obr. 10.*

1: $\forall$ *vrcholy* v *se obarví na červeno*
2: $\forall$ *vrcholy* v *se položí* $D(v) := 0$*, dále se položí* $Q := \emptyset$
3: $\forall$ *hrany* (u, v) *grafu se* $D(v)$ *zvýší o* 1
4: ***for all*** *vrchol* v*, pro který platí* $D(v) = 0$ ***do***
5: *vložit* v *do* Q
6: ***end for***
7: ***while*** $Q \neq \emptyset$ ***do***
8: *vyber libovolný* $v \in Q$ *(tedy z* Q *jej odstraň)*
9: *přebarvi* v *na zeleno*
10: ***for all*** *hrana* (v, w) ***do***
11: $D(w) - -$
12: ***if*** $D(w) = 0$ ***then***
13: *vlož* w *do* Q
14: ***end if***
15: ***end for***
16: ***end while***

Scéna (Str. 136)**.** Rychlé určení topologického uspořádání - V Algovizi zatím není implementována.

Algoritmus 13.4 (Algoritmus kritické cesty)**.** *Topologicky si uspořádáme graf (viz předchozí algoritmus). Nyní můžeme zjistit ceny nejlacinějších cest z bodu* v_0 *do všech ostatních vrcholů. Využijeme při tom dynamické programování, viz scéna níže.*

Scéna (Str. 137)**.** Algoritmus kritické cesty - Také zatím není implementována, v knížce je však algoritmus popsán.



Obrázek 12: Graf s orientovaným cyklem

Algoritmus 13.5 (Dijkstrův)**.** *Pouze pro nezáporně ohodnocené hrany. Viz [8].*

1: *Každému bodu přiřadíme* $E(v)$ *("estimation", zpočátku nastavíme* $+\infty$*), určíme množinu* M *pro dosažené body.*
2: $M = \{v_0\}$, $E(v_0) = 0$
3: ***while*** $M \neq \emptyset$ ***do***
4: $v =$ *prvek* M *s nejmenším* E
5: ***for*** $\forall(v, w)$ ***do***
6: ***if*** $E(v) == +\infty$ ***then***
7: $\{E(w) = E(v) + l(v, w); P(w) = v\}$
8: ***else if*** $E(w) > E(v) + l(v, w)$ ***then***
9: $\{E(w) = E(v) + l(v, w); P(w) = v\}$
10: ***end if***
11: ***end for***
12: *vyhoď* v *z* m
13: ***end while***

Scéna (Str. 140)**.** Dijkstrův algoritmus

Konec 11. přednášky | Chybí 12. a 13. přednáška.

Reference

[1] Domovská stránka pana profesora Kučery:
`http://kam.mff.cuni.cz/~ludek/CZ.html`
Stránka přednášky:
`http://kam.mff.cuni.cz/~ludek/NTIN060.html`

[2] Algovize je volně dostupná sada appletů od pana profesora Kučery:
`http://www.algovision.org`
Novější verze (něco je zde nové, avšak některé věci zde nefungují) je dostupná na:
`http://kam.mff.cuni.cz/~ludek/AlgovisionNew/Algovision.html`

[3] Kniha Luděk Kučera: Algovize, aneb procházka krajinou algoritmů je manuál k appletům Algovize. Lze si ji půjčit v knihovně MFF, je také ke stažení na:
`http://kam.mff.cuni.cz/~ludek/AlgovisionTexts.html`

[4] Hloubka průměrného binárního stromu (ze stránek výše):
http://kam.mff.cuni.cz/~ludek/NTIN060texty/BSTrandom.pdf

[5] Dolní odhad porovnávacího třídění (průměrý případ, ze stránek výše):
http://kam.mff.cuni.cz/~ludek/NTIN060texty/AverageLowerBoundSorting.pdf

[6] Logaritmický odhad pro obecné hashování (ze stránek výše):
http://kam.mff.cuni.cz/~ludek/NTIN060texty/hash.pdf

[7] Universální hashování (ze stránek výše):
http://kam.mff.cuni.cz/~ludek/NTIN060texty/Hash.pdf

[8] Dijkstrův algoritmus (ze stránek výše):
http://kam.mff.cuni.cz/~ludek/NTIN060texty/Dijkstra.pdf

[9] Prezentace pana docenta Čepka:
http://kti.mff.cuni.cz/~cepek/ADS1.ppt

[10] Poznámky Jana Procházky:
http://download.matfyz.info/special/algoritmy