České vysoké učení technické v Praze
Fakulta elektrotechnická

Tématické okruhy ke státní závěrečné zkoušce pro magisterský studijní program Otevřená Informatika (OI)

http://www.fel.cvut.cz/cz/education/master/topicsOI.html

OI Mgr - Společné

Vygenerováno: 9. července 2014 11:24

Obsah

1 Amortizovaná složitost. Prioritní fronty, haldy (binární, d-regulární, binomiální, Fibonacciho), operace nad nimi a jejich složitost.

Amortizovaná složitost. Amortizovaná časová složitost označuje časovou složitost algoritmu v sekvenci nejhorších možných vstupních dat. Na rozdíl od průměrné složitosti nevyužívá pravděpodobnosti a je proto zaručená [2].

Tato metoda vyžaduje znalost toho, které sekvence operací jsou vůbec možné. Nejčastěji se to týká analýzy datových struktur, které si mezi jednotlivými operacemi udržují určitý stav. Některé datové struktury mají totiž takovou vnitřní organizaci, že na ní závisí složitost, a organizovanost dat se může během posloupnosti operací měnit. Základní myšlenka amortizované analýzy tkví v tom, že operace s nejhorší složitostí změní stav struktury tak, že tento nejhorší případ nemůže nastat po dlouhý čas, tudíž amortizuje svou cenu.

Jako jednoduchý příklad můžeme uvést specifickou implementaci dynamického pole, která zdvojnásobuje velikost pole pokaždé, když dojde k jeho naplnění. V tomto případě je tedy nutná realokace, v nejhorším případě tato operace potřebuje čas až $O(n)$ - což je asymptotická složitost. Samotné vkládání prvků (bez nutnosti realokace) vyžaduje čas $O(1)$, pro n prvků tedy také $O(n)$. Pro vložení n prvků (včetně realokace) je tedy potřeba $O(n)+O(n) = O(n)$, amortizovaný čas na jedno vložení prvku je pak $O(n)/n = O(1)$ [7].

Definice. Je dána datová struktura D, na které postupně provádíme posloupnost stejných operací. Začneme s $D_0 = D$. První operace zavolaná na D_0 upraví datovou strukturu na D_1. Druhá operace zavolaná na D_1 upraví datovou strukturu na D_2. A tak dále. Postupně zavoláme i-tou operaci na D_{i-1} a ta upraví datovou strukturu na D_i. Některá operace může trvat krátce, jiná déle. Průměrný čas doby trvání operace nazveme amortizovanou časovou složitostí. Amortizovanou časovou složitost jedné operace spočítáme tak, že spočteme celkovou časovou složitost posloupnosti operací v nejhorším případě a vydělíme ji počtem operací.

K čemu je amortizovaná časová složitost? Pomůže nám lépe odhadnout časovou složitost některých algoritmů v nejhorším případě [1].

- Účetní metoda
- Metoda potenciálu

Prioritní fronta *(Priority Queue)*

Prioritní fronta je *abstraktní datový typ*, podobný klasické frontě či zásobníku s tím rozdílem, že každý element má svou "prioritu". V prioritní frontě je element s vyšší prioritou vybrán dříve než element s nižší prioritou. Pokud dva elementy mají stejnou prioritu, vyberou se v pořadí v jakém byly vloženy.

Operace. Prioritní fronta musí implementovat alespoň následující operace:

- `void push(Element e)` - vloží element do prioritní fronty
- `Element pull()` - vybere z fronty element s nejvyšší prioritou

Haldy

Halda obecně je datová struktura (obvykle stromová) splňující **vlastnost haldy**:

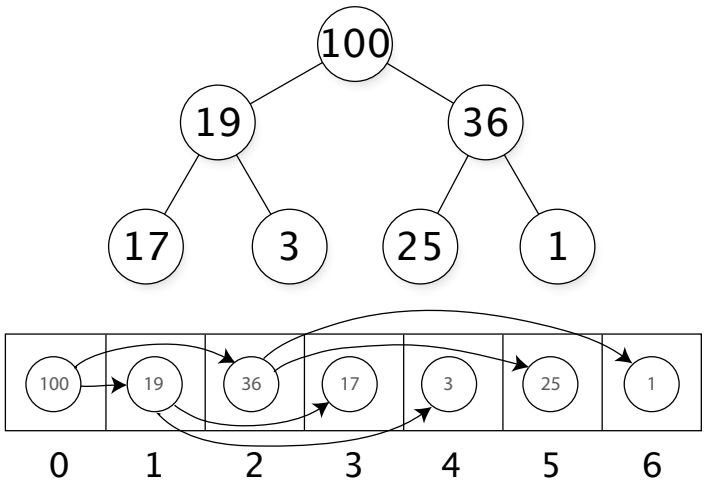
Pokud A je potomek B, pak $B \geq A$

Binární halda (Binary heap)

Binární halda je binární strom s dvěma dalšími omezeními:

1. Je to kompletní binární strom krom posledního patra (nemusí být úplné). Elementy posledního patra se plní zleva doprava.
2. Každý element je menší nebo roven vůči jeho potomkům (vlastnost haldy).

Je to jedna z možných implementací prioritní fronty. Reprezentovat ji můžeme pomocí pole (obr. 1), kde prvek na indexu i má potomky na indexech $2i$ a $2i+1$ a rodiče na indexu $i/2$.



Obrázek 1: Reprezentace (maximální) binární haldy v poli

Operace. Operace binární haldy a jejich složitosti:

- `accessMin()` - vrátí hodnotu kořene stromu (typicky první prvek pole)
- `deleteMin(e)` - vrátí element `e`, který je kořenem stromu, na jeho místo vloží nejpravější prvek `y` ze spodního patra a poté, pokud je `y` menší než jeho nejmenší potomek, prohazuji `y` s jeho nejmenším potomkem do té doby, dokud je `y` menší než jeho nově vzniknuvší nejmenší potomek, (tzv. `y` probublává stromem dolů).
- `insert(e)` - přidáme element `e` na konec haldy a dokud je předek větší než `e`, tak je prohazujeme (probublávání směrem nahoru).
- `delete(e)` - podobně jako `deleteMin()`, odeberu element `e` a nechám ho probublat
- `merge(h1, h2)` - sloučí 2 haldy, vytvoří nové pole, kam nakopíruje obsah obou hald a na toto nové pole zavolá proceduru `heapify()`. Ta jede od půlky pole směrem na začátek (navštíví všechny podhaldy) a každý vrchol podhaldy nechá probublat a správně zařadit.

- `decreaseKey(k, v)` - zmenšíme hodnotu elementu s klíčem `k` o `v` a necháme probublat stromem.

operace	čas. složitost	poznámka
`accessMin()`	$\Theta(1)$	přístup k vrcholu haldy
`deleteMin(e)`	$\Theta(\log(n))$	smazání vrcholu haldy
`insert(e)`	$\Theta(\log(n))$	přidání prvku do haldy
`delete(e)`	$\Theta(\log(n))$	smazání elementu haldy
`merge(h1,h2)`	$\Theta(n_1 + n_2)$	sloučení 2 hald
`decreaseKey(k,v)`	$\Theta(\log(n))$	snížení hodnoty klíče k o v

Tabulka 1: Binární halda - Operace a jejich složitosti

D-regulární halda (D-ary heap)

D-regulární halda je zobecnění binární haldy, kde počet potomků se rovná číslu d, namísto 2 jako je tomu v haldě binární. D nám udává počet štěpení stromu haldy. Operace jsou identické jako v binární haldě. Časová složitost operací je téměř stejná, liší se jen v základu logaritmu (u binární haldy je základ 2, tady d). Pro efektivní implementaci je vhodné zvolit d jako mocninu 2. V tomto případě lze totiž využít bitových posunů - změna indexu při průchodu polem [5].

Operace. Operace d-regulární haldy a jejich složitosti:

operace	čas. složitost	poznámka
`accessMin()`	$\Theta(1)$	přístup k vrcholu haldy
`deleteMin(e)`	$\Theta(\log_d(n))$	smazání vrcholu haldy
`insert(e)`	$\Theta(\log_d(n))$	přidání prvku do haldy
`delete(e)`	$\Theta(\log_d(n))$	smazání elementu haldy
`merge(h1,h2)`	$\Theta(n_1 + n_2)$	sloučení 2 hald
`decreaseKey(k,v)`	$\Theta(\log_d(n))$	snížení hodnoty klíče k o v

Tabulka 2: D-regulární halda - Operace a jejich složitosti

Binomiální halda (Binomial heap)

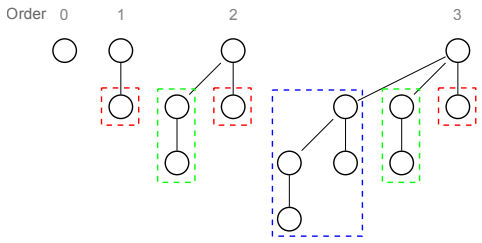
Binomiální halda je kolekce binomiálních stromů stupňů $i = 0 \dots \lfloor \log(n) \rfloor$. Každý řád je zastoupen maximálně 1 stromem.

Binomiální strom je definovám rekurzivně:

- Binomiální strom řádu 0 obsahuje jediný prvek - kořen
- Binomiální strom řádu k má kořenový element, jehož potomci jsou kořeny binomiálních stromů stupňů $k-1, k-2, \dots, 2, 1, 0$ (v tomto pořadí)

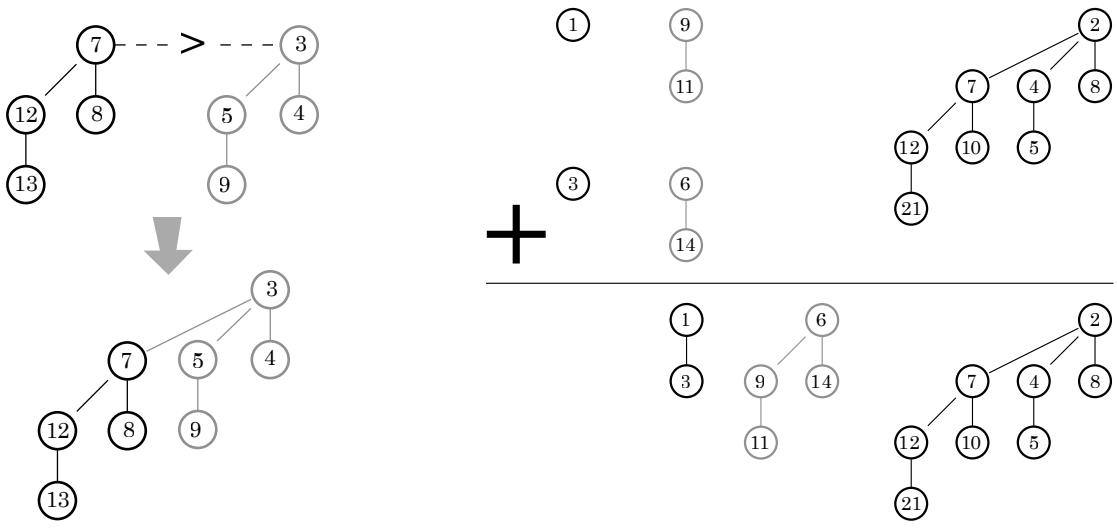
Pro binomiální strom řádu k platí:

- splňuje vlastnost haldy
- hloubka stromu je k
- kořen má k potomků
- obsahuje 2^k prvků



Operace. Operace binomiální haldy a jejich složitosti:

- `accessMin()` - vrátí kořen binomiálního stromu z MIN ukazatele.
- `deleteMin(e)` - vezmou se všechny podstromy, které vznikly odebráním kořene a ty se postupně mergují s ostatními stromy
- `insert(e)` - z vkládaného prvku se vytvoří nová binomiální halda (strom řádu 0) a ta se merguje s původní haldou
- `delete(e)` - sníží se hodnota pomocí `decreaseKey` na $-\infty$ a provede se `deleteMin()`
- `merge(h1, h2)` - Analogie mezi mergeováním dvou hald a binárním sčítáním. Nasklá-dáme si stromy obou hald pod sebe (podle jejich stupňů). Pokud v jedné haldě je strom i-tého řádu a v druhé není, tak ho jen opíši. Pokud v obou haldách existují stromy stejného řadu, pak vzniká přenos do vyššího řádu. Kdykoliv vznikne přenos, merguje-me tyto dva stromy do sebe. Díky struktuře stromů, se provádí merge, porovnáním jejich kořenů, menší z kořenů se stane kořenem nově vzniklého stromu (řádu o 1 vyšší) a druhý strom se stane jeho potomkem.
- `decreaseKey(k, v)` - podobné jako u binární haldy



Obrázek 2: Binomiální halda MERGE - 2 příklady

operace	čas. složitost	poznámka
`accessMin()`	$\Theta(1)$	přístup k vrcholu haldy
`deleteMin(e)`	$\Theta(\log(n))$	smazání vrcholu haldy
`insert(e)`	$\Theta(\log(n))$ amortizovaně: $\Theta(1)$	přidání prvku do haldy
`delete(e)`	$\Theta(\log(n))$	smazání elementu haldy
`merge(h1,h2)`	$\Theta(\log(n))$	sloučení 2 hald
`decreaseKey(k,v)`	$\Theta(\log(n))$	snížení hodnoty klíče k o v

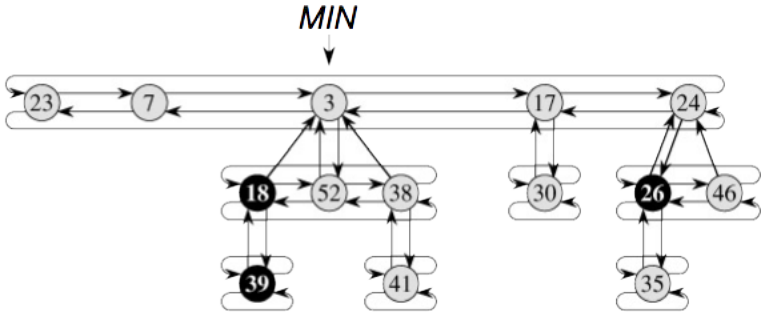
Tabulka 3: Binární halda - Operace a jejich složitosti

Fibonacciho halda

Založena na binomiální haldě. Má relaxovanější strukturu, která umožňuje zlepšené asymptotické složitosti. Fibonnaciho halda se nevyužívá v real-time systémech, protože některé operace mají lineární složitost.

Struktura. Fibonacciho haldu tvoří skupina stromů vyhovující lokální podmínce na uspořádání haldy, která vyžaduje, aby pro každý uzel stromu platilo, že prvek, který reprezentuje, je menší než prvek reprezentovaný jeho potomky. Z této podmínky vyplývá, že minimálním prvkem je vždy kořen jednoho ze stromů. Vnitřní struktura Fibonacciho haldy je v porovnání s binomiální haldou daleko více flexibilní. Jednotlivé stromy nemají pevně daný tvar a v extrémním případě může každý prvek haldy tvořit izolovaný strom nebo naopak všechny prvky mohou být součástí jediného stromu hloubky n. Tato flexibilní struktura umožňuje velmi jednoduchou implementaci operací s haldou. Operace, které nejsou potřebné, odkládáme a vykonáváme je až v okamžiku, kdy je to nevyhnutelné, například spojení nebo vložení nového prvku se jednoduše provede spojením kořenových seznamů (s konstantní náročností) a jednotlivé stromy spojíme až při operaci snížení hodnoty klíče [8].

Implementace. Pro rychlé vymazání a zřetězení se vytváří *obousměrný cyklický spojový seznam* kořenů všech stromů (obr. 3). Pro potomky každého prvku se vytváří podobný seznam. Pro každý uzel se ukládá počet synů a údaj, zda je zvýrazněn. Navíc si uchováváme ukazatel na kořenový prvek s minimální hodnotou klíče (MIN) [8].



Obrázek 3: Reprezentace fibonacciho haldy

Operace. Operace Fibonacciho haldy a jejich složitosti:

- `accessMin()` - vrátí kořen z Fibonacciho stromu na který ukazuje MIN pointer
- `deleteMin(e)` -
- `insert(e)` - vytvoří se nová halda obsahující jediný element `e`; `mark(e) = false`; Merge s původní haldou - $O(1)$
- `delete(e)` -
- `merge(h1, h2)` - prosté spojení seznamů s kořenovými prvky stromů jednotlivých hald a update MIN pointeru
- `decreaseKey(k, v)` -

Srovnání časových složitostí u hald

	binary heap	**d-ary heap**	**binomial heap**	**Fibonacci heap**
`accessMin()`	$\Theta(1)$	$\Theta(1)$	$\Theta(1)$	$\Theta(1)$
`deleteMin(e)`	$\Theta(\log(n))$	$\Theta(\log(n))$	$\Theta(\log(n))$	$O(n); O(\log(n))$
`insert(e)`	$\Theta(\log(n))$	$\Theta(\log(n))$	$O(\log(n)); O(1)$	$\Theta(1)$
`delete(e)`	$\Theta(\log(n))$	$\Theta(\log(n))$	$O(\log(n))$	$O(n); O(\log(n))$
`merge(h1,h2)`	$\Theta(n)$	$\Theta(n)$	$O(\log(n))$	$\Theta(1)$
`decreaseKey(k,v)`	$\Theta(\log(n))$	$\Theta(\log(n))$	$\Theta(\log(n))$	$O(\log(n)); O(1)$

Tabulka 4: Haldy - srovnání složitostí

2 Neorientované a orientované grafy, jejich reprezentace. Prohledávání grafu (do hloubky a do šířky), topologické uspořádání, souvislost, stromy, minimální kostra.

3 Lexikální analyzátor, syntaktický strom, syntaktický analyzátor shora dolů, LL(1) gramatiky, rozkladové tabulky.

4 Algoritmy vyhledávaní v textu s lineární a sublineární složitostí, (naivní, Boyer-Moore), využití konečných automatů pro přesné a přibližné hledání v textu.

5 Algoritmus, správnost algoritmu, složitost algoritmu, složitost úlohy, třída $\mathcal{P}$, třída $\mathcal{NP}$.

Algoritmus. *Algoritmem* rozumíme dobře definovaný proces, tj. posloupnost výpočetních kroků, který přijímá hodnoty (zadání, vstup) a vytváří hodnoty (řešení, výstup).

Řekneme, že algoritmus $\mathcal{A}$ *řeší* úlohu $\mathcal{U}$, jestliže pro každý vstup (každou instanci problému $\mathcal{U}$) vydá správné řešení.

Správnost algoritmu K ověření správnosti algoritmu je třeba ověřit 2 věci:

1. algoritmus se na každém vstupu zastaví
2. algoritmus po zastavení vydá správný výstup - řešení

Variant. Pro důkaz faktu, že se algoritmus na každém vstupu zastaví, je založen na nalezení tzv. *variantu*. Variant je hodnota udaná přirozeným číslem, která se během práce algoritmu snižuje až nabude nejmenší možnou hodnotu (a tím zaručuje ukončení algoritmu po konečně mnoha krocích).

Invariant. *Invariant*, též *podmíněná správnost algoritmu*, je tvrzení, které:

- platí před vykonáním prvního cyklu algoritmu, nebo po prvním vykonání cyklu
- platí-li před vykonáním cyklu, platí i po jeho vykonání
- při ukončení práce algoritmu zaručuje správnost řešení

Složitost algoritmu

Algoritmy lze rozdělit do několika tříd složitosti na základě času a paměti, jež potřebují ke svému vykonání na různých typech Turingových strojů. [3]

Časovou složitost algoritmu udáváme jako asymptotický odhad $T(n)$ času potřebného pro vyřešení každé instance velikosti n.

Asymptotický růst funkcí

Definujeme několik symbolů (množin).

Symbol $\mathcal{O}$. Je dána nezáporná funkce $g(n)$. Řekneme, že nezáporná funkce $f(n)$ je $\mathcal{O}(g(n))$, jestliže existuje kladná konstanta c a přirozené číslo n_0 tak, že

$$f(n) \leq c \cdot g(n) \text{ pro všechny } n \geq n_0$$

$\mathcal{O}(g(n))$ můžeme též chápat jako třídu všech nezáporných funkcí $f(n)$:

$$\mathcal{O}(g(n)) = \{f(n) \mid \exists c > 0, n_o \text{ tak, že } f(n) \leq c \cdot g(n)\ \forall n \geq n_0\}$$

Další symboly:

- $\Omega(g(n)) = \{f(n) \mid \exists c > 0, n_o$ tak, že $f(n) \geq c \cdot g(n)\ \forall n \geq n_0\}$
- $\Theta(g(n)) = \{f(n) \mid \exists c_1, c_2 > 0, n_o$ tak, že $c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n)\ \forall n \geq n_0\}$
- $o(g(n)) = \{f(n) \mid \forall c > 0\ \exists n_o$ tak, že $0 \leq f(n) < c \cdot g(n)\ \forall n \geq n_0\}$
- $\omega(g(n)) = \{f(n) \mid \forall c > 0\ \exists n_o$ tak, že $0 \leq c \cdot g(n) < f(n)\ \forall n \geq n_0\}$

Tranzitivita $\mathcal{O}, \Omega, \Theta$. Máme dány tři nezáporné funkce $f(n), g(n), h(n)$

- Jestliže $f(n) \in \mathcal{O}(g(n))$ a $g(n) \in \mathcal{O}(h(n))$, pak $f(n) \in \mathcal{O}(h(n))$
- Jestliže $f(n) \in \Omega(g(n))$ a $g(n) \in \Omega(h(n))$, pak $f(n) \in \Omega(h(n))$
- Jestliže $f(n) \in \Theta(g(n))$ a $g(n) \in \Theta(h(n))$, pak $f(n) \in \Theta(h(n))$

Reflexivita $\mathcal{O}, \Omega, \Theta$. Pro všechny nezáporné funkce $f(n)$ platí:

- $f(n) \in \mathcal{O}(f(n))$
- $f(n) \in \Omega(f(n))$
- $f(n) \in \Theta(f(n))$

Master Theorem

Používá se pro určení asymptotického časového odhadu u rekurentních vztahů.
Jsou dána přirozená čísla $a \geq 1, b \geq 1$ a funkce $f(n)$. Předpokládejme, že funkce $T(n)$ je dána na přirozených číslech rekurentním vztahem

$$T(n) = aT\left(\tfrac{n}{b}\right) + f(n), \text{ kde } \tfrac{n}{b} \text{ znamená buď } \lfloor \tfrac{n}{b} \rfloor \text{ nebo } \lceil \tfrac{n}{b} \rceil.$$

1. Jesltiže $f(n) \in \mathcal{O}(n^{\log_b a - \epsilon})$ pro nějakou konstantu $\epsilon > 0$, pak $T(n) \in \Theta(n^{\log_b a})$.
2. Jesltiže $f(n) \in \Theta(n^{\log_b a})$, pak $T(n) \in \Theta(n^{\log_b a} \lg n)$.
3. Jesltiže $f(n) \in \Omega(n^{\log_b a + \epsilon})$ pro nějakou konstantu $\epsilon > 0$ a jestliže $af(\frac{n}{b}) \leq cf(n)$ pro nějakou konstantu $c < 1$ pro všechna dostatečně velká n, pak $T(n) \in \Theta(f(n))$.

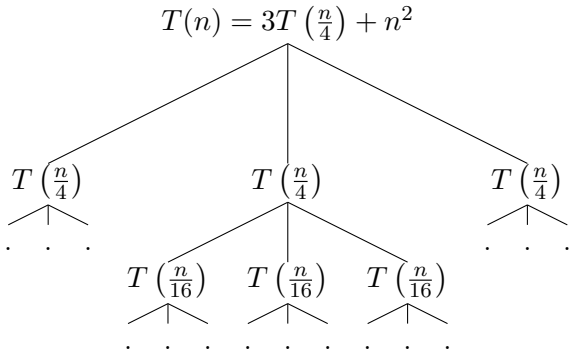
Poznámka. MT nepokrývá všechny případy.

► **Příklad:**

$T(n) = 6T\left(\frac{n}{4}\right) + n^2.\lg(n)$ // 3. případ $n^2 \lg(n) \in \Omega(n^{\log_4 6})$
$6\left(\frac{n}{4}\right)^2 \lg\left(\frac{n}{4}\right) \leq c \cdot n^2 \lg(n)$ // roznásobení
$\frac{6}{16} n^2 \lg\left(\frac{n}{4}\right) \leq c \cdot n^2 \lg(n)$ // $\lg\left(\frac{n}{4}\right)$ si "zvětším"na $\lg(n)$
$\frac{6}{16} n^2 \lg(n) \leq c \cdot n^2 \lg(n)$ // pokrátím (vydělím) $n^2 \lg(n)$
$\frac{6}{16} \leq c$ // $c < 1$, platí
$\rightarrow T(n) = \Theta(n^2 \lg(n))$

Řešení rekurzivních vztahů pomocí rekurzivních stromů

► **Příklad:**



Vytvoříme si jednotlivé hladiny stromu, který popisuje rekurzivní výpočet funkce $T(n)$. V nulté hladině máme pouze $T(n)$ a hodnotu n^2, kterou potřebujeme k výpočtu $T(n)$ (známe-li $T\left(\frac{n}{4}\right)$.

V první hladině se nám výpočet $T(n)$ rozpadl na tři výpočty $T(n)$. K tomu potřebujeme hodnotu $3 \cdot (\frac{n}{4})^2 = \frac{3}{16}n^2$.

Při přechodu z hladiny i do hladiny $i+1$ se každý vrchol rozdělí na tři a každý přispěje do celkové hodnoty jednou šestnáctinou předchozího. Proto je součet v hladině i roven $(\frac{3}{16})^i n^2$.

Poslední hladina má vrcholy označené hodnotami $T(1)$ a tím rekurze končí. Počet hladin odpovídá $\log_4 n$. V poslední hladině je $3^{\log_4 n} = n^{\log_4 3}$ hodnot $T(1)$. Proto platí

$$T(n) = \sum_{i=0}^{\log_4 n} \left(\frac{3}{16}\right)^i n^2 + \Theta(n^{\log_4 3})$$

Odtud

$$T(n) < n^2 \sum_{i=0}^{\infty} \left(\frac{3}{16}\right)^i + \Theta(n^{\log_4 3}) = n^2 \frac{1}{1-\frac{3}{16}} + \Theta(n^{\log_4 3}) = \frac{16}{13}n^2 + \Theta(n^{\log_4 3})$$

Proto $T(n) \in \Theta(n^2)$

Složitost úlohy

Složitost úlohy je složitost nejlepšího algoritmu řešícího danou úlohu.

Turingův stroj (Turing machine - TM)

Je teoretický model počítače, který se skládá z:

- z **řídící jednotky**, která se může nacházet v jednom z konečně mnoha stavů
- potenciálně **nekonečné pásky** (nekonečné na obě strany) rozdělené na jednotlivé pole
- **čtecí hlavy**, která umožňuje číst obsah polí a přepisovat obsah polí pásky

Je dán sedmicí $(Q, \Sigma, \Gamma, \delta, q_0, B, F)$, kde

- Q je konečná množina stavů
- Σ je konečná množina vstupních symbolů
- Γ je konečná množina páskových symbolů, přitom $\Sigma \subset \Gamma$
- B - je prázdný symbol (*blank*), jedná se o páskový symbol, který není vstupním symbolem (tj. $B \in \Gamma \setminus \Sigma$)
- δ je přechodová funkce, tj. parciální zobrazení z množiny $(Q \setminus F) \times \Gamma$ do množiny $Q \times \Gamma \times L, R$, (zde L znamená pohyb hlavy o jedno pole doleva, R pohyb doprava)
- $q_0 \in Q$ je počáteční stav
- $F \subset Q$ je množina koncových stavů

Nedeterministický TM Je takový Turingův stroj, u kterého připustíme, aby v jedné situaci mohl provézt několik různých kroků.

Přijímaný a rozhodovaný jazyk TM. Vstupní slovo $w \in \Sigma^*$ je *přijato* Turingovým strojem M právě tehdy, když se Turingův stroj na slově w úspěšně zastaví. Množinu slov $w \in \Sigma^*$, které Turingův stroj přijímá, se nazývá *jazyk přijímaný* M a značíme ho $L(M)$.

Turingův stroj *rozhoduje* jazyk L, jestliže tento jazyk přijímá a navíc se na každém vstupu zastaví.

Třída složitosti - $\mathcal{P}$

Třída $\mathcal{P}$ Řekneme, že rozhodovací úloha $\mathcal{U}$ leží ve třídě $\mathcal{P}$, jestliže **existuje deterministický** Turingův stroj, který **rozhodne** jazyk $L_{\mathcal{U}}$ a pracuje v **polynomiálním čase**; tj. funkce $T(n)$ je $\mathcal{O}(p(n))$ pro nějaký polynom $p(n)$.

Příklady $\mathcal{P}$ úloh:

- **Minimální kostra v grafu**. Je dán neorinetovaný graf G s ohodnocením hran c. Je dáno číslo k. Existuje kostra grafu ceny menší nebo rovno k?
- **Nejkratší cesty v acyklickém grafu**. Je dán acyklický graf s ohodnocením hran a. Jsou dány vrcholy r a c. Je dáno číslo k. Existuje orientovaná cesta z vrcholu r do vrcholu c délky menší nebo rovno k?

- **Toky v sítích**. Je dána síť s horním omezením c, dolním omezením l, se zdrojem z a spotřebičem s. Dále je dáno číslo k. Existuje přípustný tok od z do s velikosti alespoň k?
- **Minimální řez**. Je dána síť s horním omezením c, dolním omezením l. Dále je dáno číslo k. Existuje řez, který má kapacitu menší nebo rovnu k?

Třída složitosti - $\mathcal{NP}$

Třída $\mathcal{NP}$ Řekneme, že rozhodovací úloha $\mathcal{U}$ leží ve třídě $\mathcal{NP}$, jestliže **existuje nedeterministický** Turingův stroj, který **rozhodne** jazyk $L_{\mathcal{U}}$ a pracuje v **polynomiálním čase**.

Příklady $\mathcal{NP}$ úloh:

- **Kliky v grafu**. Je dán neorinetovaný graf G a číslo k. Existuje klika v grafu G o alespoň k vrcholech?
- **Nejkratší cesty v obecném grafu**. Je dán orientovaný graf s ohodnocením hran a. Jsou dány vrcholy r a v. Je dáno číslo k. Existuje orientovaná cesta z vrcholu r do vrcholu c délky menší nebo rovno k?
- **k-barevnost**. Je dán neorientovaný graf G. Je graf G k-barevný?
- **Knapsack**. Je dáno n předmětů $1, 2, \ldots, n$. Každý předmět i má cenu c_i a váhu w_i. Dále jsou dána čísla A a B. Je možné vybrat předměty tak, aby celková váha nepřevýšila A a celková cena byla alespoň B?

Otázka obsahuje texty, úryvky a definice z [6].

6 $\mathcal{NP}$-úplné a $\mathcal{NP}$-těžké úlohy, Cookova věta, heuristiky na řešení $\mathcal{NP}$-těžkých úloh, pravděpodobnostní algoritmy.

Než nadefinujeme třídu $\mathcal{NPC}$, musíme definovat (polynomiální) *redukci úloh.*

Redukce a polynomiální redukce úloh. Jsou dány dvě rozhodovací úlohy $\mathcal{U}$ a $\mathcal{V}$. Řekneme, že úloha $\mathcal{U}$ se *redukuje* na úlohu $\mathcal{V}$, jestliže existuje algoritmus (program pro RAM, Turingův stroj) M, který pro každou instanci I úlohy $\mathcal{U}$ zkonstruuje instanci I' úlohy $\mathcal{V}$ a to tak, že

$$I \text{ je ANO-instance } \mathcal{U} \text{ iff } I' \text{ je ANO-instance } \mathcal{V}$$

Fakt, že úloha $\mathcal{U}$ se redukuje na úlohu $\mathcal{V}$ značíme

$$\mathcal{U} \lhd \mathcal{V}.$$

Jestliže navíc, algoritmus M pracuje v polynomiálním čase, říkáme, že $\mathcal{U}$ se *polynomiálně* redukuje na $\mathcal{V}$ a značíme

$$\mathcal{U} \lhd_p \mathcal{V}.$$

Třída složitosti - $\mathcal{NPC}$ ($\mathcal{NP}$-complete, $\mathcal{NP}$-úplná)

$\mathcal{NP}$ úplné úlohy. Řekneme, že rozhodovací úloha $\mathcal{U}$ je $\mathcal{NP}$ *úplná* , jestliže

1. $\mathcal{U}$ je ve třídě $\mathcal{NP}$
2. každá $\mathcal{NP}$ úloha se polynomiálně redukuje na $\mathcal{U}$.

Příklady $\mathcal{NPC}$ úloh:

- **SAT** Splnitelnost formulí v konjunktivním normálním tvaru.
- **3 - CNF SAT**
- **3-barevnost**
- **ILP**

Třída složitosti - $\mathcal{NP}$-hard ($\mathcal{NP}$-těžká)

$\mathcal{NP}$ obtížné úlohy. Jestliže o některé úloze $\mathcal{U}$ pouze víme, že se na ní polynomiálně redukuje některá $\mathcal{NP}$ úplná úloha, pak říkáme, že $\mathcal{U}$ je $\mathcal{NP}$ těžká, nebo též $\mathcal{NP}$ obtížná. Poznamenejme, že to vlastně znamená, že $\mathcal{U}$ je alespoň tak těžká jako všechny $\mathcal{NP}$ úlohy.

Cookova věta

Dle Cookovy věty lze převést v polynomiálním čase libovolný nedeterministický Turingův stroj na problém splnitelnosti booleovských formulí v konjunktivním normálním tvaru (CNF SAT).

Důsledkem této věty je vymezení skupiny úloh, které jsou nejtěžší v rámci všech problémů třídy NP. O těchto úlohách, na které lze převést v polynomiálním čase libovolnou jinou úlohu z NP, říkáme, že jsou $\mathcal{NP}$-úplné ($\mathcal{NP}$-complete).

Důkaz. Není těžké se přesvědčit že úloha SAT je ve třídě $\mathcal{NP}$. První fáze nedeterministického algoritmu vygeneruje ohodnocení logických proměnných a na základě tohoto ohodnocení jsme schopni v polynomiálním čase ověřit, zda je v tomto ohodnocení formule pravdivá nebo ne.

Druhá část důkazu spočívá v popisu práce TM formulí výrokové logiky. Načrtneme si základní myšlenku tohoto popisu.

Je dán NTM $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$. Předpokládejme, že M přijímá slovo w a potřebuje při tom $p(n)$ kroků.

Zavedeme logické proměnné:

- $h_{i,j},\ i = 0, 1, \ldots, p(n); j = 1, 2, \ldots, p(n);$
  - $h_{i,j}$ je rovna 1, pokud hlava TM v čase i čte j-té pole pásky.
- $s_i^q,\ i = 0, 1, \ldots, p(n); q \in Q$
  - s_i^q je rovna 1, pokud TM v čase i je ve stavu q.
- $t_{i,j}^A,\ i = 0, 1, \ldots, p(n); j = 1, 2, \ldots, p(n); A \in \Gamma$
  - $t_{i,j}^A$ je rovna 1, pokud v čase i v j-té poli pásky je páskový symbol A.

Nyní je třeba formulemi popsat následující fakta:

1. V každém okamžiku je TM v právě jednom stavu.
2. V každém okamžiku čte hlava TM právě jedno pole vstupní pásky.
3. V každém okamžiku je na každém poli pásky TM právě jeden páskový symbol.
4. Na začátku práce (tj. v čase 0) je TM ve stavu q_0, hlava čte první pole pásky a na pásce je na prvních n polích vstupní slovo, ostatní pole pásky obsahují B.
5. Krok TM je určen přechodovou funkcí, tj. stav stroje, obsah čteného pole a poloha hlavy v čase $i + 1$ je dána přechodovou funkcí.
6. V polích pásky, které v čase i hlava nečte, je obsah v čase $i + 1$ stejný jako v i.
7. Na konci práce TM , tj. v čase $p(n)$, je stroj ve stavu q_f.

Ukážeme jak utvořit formule pro body **1**, **4**, **5**, **6** a **7**.

Bod **1**. V okamžiku i je TM v aspoň jednom stavu:

$$\bigvee_{q \in Q} s_i^q.$$

V okamžiku i TM není ve dvou různých stavech:

$$\bigwedge_{q \neq q'} (\neg s_i^q \vee \neg s_i^{q'}).$$

Nyní fakt, že TM je v okamžiku i právě v jednom stavu je konjunkce obou výše uvedených formulí:

$$(\bigvee_{q \in Q} s_i^q) \wedge \bigwedge_{q \neq q'} (\neg s_i^q \vee \neg s_i^{q'}).$$

Bod **4**. Na začátku práce (tj. v čase 0) je TM ve stavu q_0, hlava čte první pole pásky a na pásce je na prvních n polích vstupní slovo $a_1 a_2 \dots a_n$, ostatní pole pásky obsahují B.

$$s_0^{q_0} \wedge h_{0,1} \wedge t_{0,1}^{a_1} \wedge \dots \wedge t_{0,n}^{a_n} \wedge t_{0,n+1}^{B} \wedge \dots \wedge t_{0,p(n)}^{B}.$$

Bod **5**. Jestliže TM je v čase i ve stavu q, hlava je na j-tém poli pásky, hlava čte páskový symbol A a $\delta(q, A)$ se skládá z trojic (p, C, D) (zde $D = 1$ znamená posun hlavy doprava, $D = -1$ znamená posun hlavy doleva), pak formule má tvar:

$$\bigwedge_j \bigwedge_{A \in \Gamma} ((s_i^q \wedge h_{i,j} \wedge t_{i,j}^A) \Rightarrow \bigvee (s_{i+1}^p \wedge t_{i+1,j}^C \wedge h_{i+1,j+D})).$$

Bod **6**. Obsah polí kromě j-tého zůstává v čase $i + 1$ stejný:

$$\bigwedge_j \bigwedge_{A \in \Gamma} ((\neg h_{i,j} \wedge t_{i,j}^A) \Rightarrow t_{i+1,j}^A).$$

Bod **7**. Na konci práce TM, tj. v čase $p(n)$ je stroj ve stavu q_f:

$$s_{p(n)}^{g_f}.$$

Výslednou formuli dostaneme jako konjunkci všech dílčích formulí pro všechny časové okamžiky $i = 0, 1, \dots, p(n)$.

Heuristiky na řešení $\mathcal{NP}$-těžkých úloh

Jestliže je třeba řešit problém, který je $\mathcal{NP}$ úplný, musíme pro větší instance opustit myšlenku přesného nebo optimálního řešení a smířit se s tím, že získáme "dostatečně přesné"nebo "dostatečně kvalitní"řešení. K tomu se používají heuristické algoritmy pracující v polynomiálním čase. Algoritmům, kde umíme zaručit "jak daleko"je nalezené řešení od optimálního, se také říká aproximační algoritmy.

Trojúhelníková nerovnost. Řekneme, že instance obchodního cestujícího splňuje trojúhelníkovou nerovnost, jestliže pro každá tři města i, j, k platí:

$$d(i,j) \leq d(i,k) + d(k,j).$$

2-aproximační algoritmus

Jestliže instance I obchodního cestujícího splňuje trojúhelníkovou nerovnost, pak existuje polynomiální algoritmus $\mathcal{A}$, který pro I najde trasu délky D, kde $D \leq 2OPT(I)$ ($OPT(I)$ je délka optimální trasy v I).

Slovní popis algoritmu. Instanci I považujeme za úplný graf G s množinou vrcholů $V = \{1, 2, \ldots, n\}$ a ohodnocením d.

1. V grafu G najdeme minimální kostru (V, K).
2. Kostru (V, K) prohledáme do hloubky z libovolného vrcholu.
3. Trasu T vytvoříme tak, že vrcholy procházíme ve stejném pořadí jako při prvním navštívení během prohledávání grafu. T je výstupem algoritmu.

Zřejmě platí, že délka kostry K je menší než $OPT(I)$. Ano, vynecháme- li z optimální trasy některou hranu, dostaneme kostru grafu G. Protože K je minimální kostra, musí být délka K menší než $OPT(I)$ (předpokládáme, že vzdálenosti měst jsou kladné). Vzhledem k platnosti trojúhelníkové nerovnosti, je délka T menší nebo rovna dvojnásobku délky kostry K.

Christofidesův algoritmus

Jestliže instance I obchodního cestujícího splňuje trojúhelníkovou nerovnost, pak následující algoritmus najde trasu T délky D takovou, že $D \leq \frac{3}{2}OPT(I)$.

Instanci I považujeme za úplný graf G s množinou vrcholů $V = \{1, 2, \ldots, n\}$ a ohodnocením d.

1. V grafu G najdeme minimální kostru (V, K).
2. Vytvoříme úplný graf H na množině všech vrcholů, které v kostře (V, K) mají lichý stupeň.
3. V grafu H najdeme nejlevnější perfektní párování P[1].
4. Hrany P přidáme k hranám K minimální kostry. Graf $(V, P \cup K)$ je eulerovský graf. V grafu $(V, P \cup K)$ sestrojíme uzavřený eulerovský tah.
5. Trasu T získáme z eulerovského tahu tak, že vrcholy navštívíme v pořadí, ve kterém jsme do nich poprvé vstoupili při tvorbě eulerovského tahu.

Platí, že délka takto vzniklé trasy je maximálně $\frac{3}{2}$ krát větší než délka optimální trasy.

[1] Párování grafu je v teorii grafů taková podmnožina hran grafu, že žádné dvě hrany z této množiny nemají společný vrchol. (Idea je taková, že vrcholy grafu dáváme do párů. Pár může vzniknout jen tam, kde byla hrana. Přitom každý vrchol může být jen v jednom páru.) Perfektní párování grafu je párování, které pokrývá všechny vrcholy grafu [9].

Pravděpodobnostní algoritmy

Randomizovaný Turingův stroj (RTM). RTM je, zhruba řečeno, Turingův stroj M se dvěma nebo více páskami (pásky > 2 obsahují B), kde první páska má stejnou roli jako u deterministického Turingova stroje, ale druhá páska obsahuje náhodnou posloupnost 0 a 1, tj. na každém políčku se 0 objeví s pravděpodobností $\frac{1}{2}$ a 1také s pravděpodobností $\frac{1}{2}$.

Třída $\mathcal{RP}$. Jazyk L patří do třídy $\mathcal{RP}$ právě tehdy, když existuje RTM M takový, že:

1. Jestliže $w \notin L$, stroj M se ve stavu q_f zastaví s pravděpodobností 0.
2. Jestliže $w \in L$, stroj M se ve stavu q_f zastaví s pravděpodobností, která je alespoň rovna $\frac{1}{2}$.
3. Existuje polynom $p(n)$ takový, že každý běh M (tj. pro jakýkoli obsah druhé pásky) trvá maximálně $p(n)$ kroků, kde n je délka vstupního slova.

Příklady $\mathcal{RP}$ úloh:

- **Miller-Rabinův test prvočíselnosti**

TM typu Monte-Carlo. RTM splňující podmínky 1 a 2 z definice $\mathcal{RP}$ se nazývá TM typu *Monte-Carlo* (obecně nemusí pracovat v polynomiálním čase).

Třída $\mathcal{ZPP}$

Jazyk L patří do třídy $\mathcal{ZPP}$ právě tehdy, když existuje RTM M takový, že:

1. Jestliže $w \notin L$, stroj M se ve stavu q_f zastaví s pravděpodobností 0.
2. Jestliže $w \in L$, stroj M se ve stavu q_f zastaví s pravděpodobností 1.
3. Střední hodnota počtu kroků M v jednom běhu je $p(n)$, kde $p(n)$ je polynom a n je délka vstupního slova.

To znamená: M neudělá chybu, ale nezaručujeme vždy polynomiální počet kroků při jednom běhu, pouze střední hodnota počtu kroků je polynomiální.

TM typu Las-Vegas. RTM splňující podmínky z definice $\mathcal{ZPP}$ se nazývá TM typu *Las-Vegas*.

Otázka obsahuje texty, úryvky a definice z [6].

7 Metoda větví a mezí. Algoritmy pro celočíselné lineární programování. Formulace optimalizačních a rozhodovacích problémů pomocí celočíselného lineárního programování. Toky a řezy. Multi-komoditní toky.

ILP - celočíselné lineární programování

Úloha celočíselného lineárního programování (LP) je zadána maticí $\mathbf{A} \in \mathbf{R}^{m \times n}$ a vektory $b \in \mathbf{R}^m, c \in \mathbf{R}^n$. Cílem je najít takový vektor $x \in \mathbf{Z}^n$, že platí $\mathbf{A} \cdot x \leq b$ a $c^T \cdot x$ je maximální.

Obvykle se celočíselné lineární programování zapisuje ve tvaru:

$$max(c^T \cdot x : \mathbf{A} \cdot x \leq b, x \in Z^n)$$

Pokud bychom takovou úlohu řešili pomocí lineárního programování s tím, že bychom výsledek zaokrouhlili, nejenom že bychom neměli zaručeno že výsledné řešení bude optimální ale ani to, zda bude přípustné. Zatímco úloha LP je řešitelná v polynomiálním čase, úloha ILP je tzv. NP-těžká (NP-hard), neboli není znám algoritmus, který by vyřešil libovolnou instanci této úlohy v polynomiálním čase. Protože prostor řešení ILP není konvexní množina, nelze přímo aplikovat metody konvexní optimalizace.

Formulace optimalizačních a rozhodovacích problémů pomocí ILP.

Algoritmy pro celočíselné lineární programování

1. Výčtové metody (Enumerative Methods)
2. Metoda větví a mezí (Branch and Bound)
3. Metody sečných nadrovin (Cutting Planes Methods)

Výčtové metody (Enumerative Methods)

Výpočet je založen na prohledávání oblasti zahrnující všechna přípustná řešení [1, 2]. Vzhledem k celočíselnému omezení proměnných je počet těchto řešení konečný ale jejich počet je extrémně vysoký. Proto je tato metoda vhodná pouze pro malé problémy s omezeným počtem diskrétních proměnných. Postup je možno zobecnit na úlohu MIP (mixed IP) tak, že ke každé kombinaci diskrétních proměnných je vyřešena úloha LP kde jsou diskrétní proměnné považovány za konstanty. [4]

Metoda větví a mezí (Branch and Bound)

text + obr s prikladem

Metody sečných nadrovin (Cutting Planes Methods)

Další skupinou algoritmů jsou metody sečných nadrovin (cutting plane method), založené podobně jako metoda větví a mezí na opakovaném řešení úlohy LP. Výpočet je prováděn iterativně tak, že v každém kroku je přidána další omezující podmínka zužující oblast přípustných řešení. Každá nová omezující podmínka musí splňovat tyto vlastnosti:

1. Optimální řešení nalezené pomocí LP se stane nepřípustným.
2. Žádné celočíselné řešení přípustné v předchozím kroku se nesmí stát nepřípustným.

Nové omezení splňující tyto vlastnosti je přidáno v každé iteraci. Vzniklý ILP program je vždy znovu řešen jako úloha LP. Proces je opakován, dokud není nalezeno přípustné celočíselné řešení. Konvergence takovéhoto algoritmu potom závisí na způsobu přidávání omezujících podmínek. Mezi nejznámější metody patří Dantzigovi řezy (Dantzig cuts) a Gomoryho řezy (Gomory cuts). [4]

Formulace optimalizačních a rozhodovacích problémů pomocí celočíselného lineárního programování

Toky a řezy

Multi-komoditní toky

8 Nejkratší cesty. Úloha obchodního cestujícího. Heuristiky a aproximační algoritmy. Metoda dynamického programování. Problém batohu. Pseudo-polynomiální algoritmy.

1. SPT - definovat, spousta problemu se na to da prevezt, trojuhelnikova nerovnost
2. TSP - definovat, slozitost, dukaz NP z redukce z Ham Cycle + dukaz neexistence aprox alg, heuristiky 2 aprox atd
3. Knapsack - definovat, dynamicke programovani, 2aprox, pseudo polym alg

9 Rozvrhování na jednom procesoru a na paralelních procesorech. Rozvrhování projektu s časovými omezeními. Programování s omezujícími podmínkami.

1. klasicke rozvhovani, multi, atd.
2. list scheduling
3. vsechny ty algoritmy

Reference

[1] Ing. Jakub Černý, PhD. Amortizovaná časová složitost. Dostupné z: http://algoritmy.eu/zga/casova-slozitost/amortizovana/.

[2] Ing. Pavel Mička. Amortizovaná složitost, . Dostupné z: http://www.algoritmy.net/article/3024/Amortizovana-slozitost.

[3] Ing. Pavel Mička. Třídy složitosti a Turingovy stroje, . Dostupné z: http://www.algoritmy.net/article/5774/Tridy-slozitosti.

[4] Ing. Přemysl Šůcha, Ph.D. *Celočíselné lineární programování* [online]. 2004. Dostupné z: http://support.dce.felk.cvut.cz/pub/hanzalek/_private/ref/sucha_ilp.pdf.

[5] Jiří Vyskočil, Radek, Mařík, Marko Berezovský. *Slidy k přednáškám PAL* [online]. 2013. Dostupné z: https://cw.felk.cvut.cz/wiki/courses/a4m33pal/prednasky.

[6] Prof. RNDr. Marie Demlová, CSc. *Slidy k přednáškám TAL* [online]. 2014. Dostupné z: http://math.feld.cvut.cz/demlova/teaching/tal/predn_tal.html.

[7] Přispěvovatelé wikipedie. Asymptotická složitost, . Dostupné z: http://cs.wikipedia.org/wiki/Asymptotickásložitost.

[8] Přispěvovatelé wikipedie. Fibonacciho halda, . Dostupné z: http://cs.wikipedia.org/wiki/Fibonacciho_halda.

[9] Přispěvovatelé wikipedie. Párování grafu, . Dostupné z: http://cs.wikipedia.org/wiki/Párování_grafu.