2. Složitost, grafové algoritmy *(zapsal Martin Koutecký)*

Model RAM

Při analýze algoritmu bychom chtěli nějak popsat jeho složitost. Abychom mohli udělat toto, potřebujeme nejprve definovat výpočetní model. Výpočetních modelů je více, my vybereme jeden poměrně blízký skutečným počítačům:

Definice: Random Access Machine (RAM)

RAM počítá jen s celými čísly – znaky, stringy a podobně reprezentujeme čísly, jejich posloupnostmi atd. Paměť je tvořena buňkami, které obsahují čísla. Paměťové buňky jsou adresované taktéž čísly. A program samotný je konečná posloupnost instrukcí následujících druhů:

- Aritmetické a logické: X `<-` $Y \oplus Z, \oplus \in \{+, -, *, \texttt{div}, \texttt{mod}, \&, |, <<, >>\}$
- Řídící: `goto` *label*, `halt`
- Podmínky: pro libovolnou nepodmíněnou instrukci můžu použít if X `<` Y `==>` instrukce

Poznámka (operandy):

- Konstanty (1, 2, ...)
- Adresované přímo – M[konst.] – budeme používat písmena `A-Z` jako aliasy pro buňky paměti -1 až -26, které nazýváme registry. (tedy A=`M[-1]`)
- Adresované nepřímo – `M[M[konst.]]` – budeme používat zkratku [[konst.]]

Samotný výpočet probíhá takto:

1. Do smluvených buněk umístíme vstup, obsah zbylých paměťových buněk není definován.
2. Provádíme program postupně po instrukcích, dokud nedojdeme k haltu nebo konci programu.
3. Pokud se program nezacyklil, tedy pokud skončil, ze smluvených buněk přečteme výstup.

Složitost

Jak dobře popsat složitost?

1. RAM *s jednotkovou cenou*: čas $\approx$ #instrukcí, prostor $\approx$ maximální číslo buňky minus minimální číslo buňky použité při výpočtu. Toto není moc dobrý nápad, protože není nijak penalizována například práce s velmi dlouhými čísly – pořád je to jedna instrukce, takže cena je stejná, ale počítače se tak přece nechovají. Velikost čísel ale omezit nesmíme, protože bychom omezili paměť (čísly ji adresujeme).

2. RAM *s logaritmickou cenou*: cena instrukce $\approx$ #bitů zpracovávaných čísel, prostor $\approx$ # bitů všech použitých buněk. To je teoreticky přesné, ale dost nepraktické (ve všech složitostech by byly logaritmy).
3. RAM *s omezenými čísly*: jednotková cena instrukcí, ale čísla omezíme nějakým polynomem $P(n)$. Tím zmizí paradoxy prvního modelu, ale můžeme adresovat jen polynomiální prostor (to nám ovšem obvykle nevadí).

Nadále tedy budeme předpokládat třetí zmíněný model.

Definice:

- *Čas běhu algoritmu* $t(x)$ pro vstup x měříme jako sumu časů provedených operací, které program provedl při zpracování vstupu x.
- *Prostor běhu algoritmu* $s(x)$ je analogicky počet paměťových buněk spotřebovaných při výpočtu se vstupem x.
- *Časová složitost* (v nejhorším případě) je:

$$T(n) := \max\{t(x); x \text{ je vstup délky } n\}.$$

- *Prostorová složitost* (v nejhorším případě) je:

$$S(n) := \max\{s(x); x \text{ je vstup délky } n\}.$$

Nyní zkusíme zanalyzovat nějaký konkrétní algoritmus. Vezměme například řazení pomocí přímeho výběru (selection sort). Na vstupu dostaneme počet čísel n (v registru `N`), v buňkách $1, \ldots, n$ je nesetříděná posloupnost čísel. Ta pak třídíme následujícím algoritmem zapsaným v pseudokódu:

1. Pro $i = 1$ do n:
2. $\quad j \leftarrow i$
3. $\quad$ Pro $k = i$ do n:
4. $\quad\quad$ Je-li $[k] < [j] \Rightarrow j \leftarrow k$
5. $\quad [i]$ prohodíme s $[j]$.

Jak by takový algoritmus vypadal zapsaný v instrukcích RAM? Budeme muset použít návěští a goto místo cyklů, jména registrů místo proměnných a třeba prohození musíme provést přes třetí proměnnou. Nějak takto:

```
        I <- 1
LOOP:   J <- I
        M <- I
MIN:    IF [J]>=[M] ==> GOTO NEXT
        M <- J
NEXT:   J <- J+1
        IF J<=N ==> GOTO MIN
        X <- [I]
```

```
[I] <- [M]
[M] <- X
I <- I+1
IF I<=N ==> GOTO LOOP
```

Pojďme se podívat, jaká je časová složitost jednotlivých částí algoritmu. Cyklus `MIN` provede za průchod 3 nebo 4 instrukce, ale zajímá nás nejhorší případ, takže 4. Zavolá se $(N - I + 1)$-krát, tedy celkem provede $4 \cdot (N - I + 1)$ instrukcí.

Mimo cyklu `MIN` je v `LOOP` ještě 7 instrukcí, tedy celý `LOOP` provede $4 \cdot (N - I + 1) + 7 = 4(N - 1) + 11$ instrukcí.

Celkově se dostáváme k součtu

$$1 + (\sum_{I=1}^{N} 4(N - I) + 11) = 1 + 11N + 4 \cdot \frac{N(N-1)}{2} = 2N^2 + 9N + 1.$$

Na multiplikativních konstantách ale nezáleží – Na reálných strojích se ceny jednotlivých (pro nás jednotkových) instrukcí stejně liší, takže nemá cenu se multiplikativními konstantami zabývat (alespoň při prvním přiblížení k problému).

$$2N^2 + 9N + 1 \approx N^2 + N$$

Navíc asymptoticky pomalejší funkce nakonec pro velké N vždy prohraje. Tím pádem nezáleží ani na členech nižších řádů:

$$N^2 + N \approx N^2$$

Když už toto víme, můžeme zanedbávat konstanty průběžně: N cyklů po $\approx N$ krocích $\Rightarrow \approx N^2$ kroků. To nás vede k zavedení tzv. *asymptotické notace:*

Asymptotická notace

Definice: Pro funkce $f, g : \mathbb{N} \to \mathbb{R}^+$ řekneme, že f je $\mathcal{O}(g)$ právě tehdy, když $\exists c > 0 : \forall^* n \in \mathbb{N} : f(n) \leq c \cdot g(n)$. Zde $\forall^* n \in \mathbb{N}$ je zkratka za „$\exists n_0 \in \mathbb{N} : \forall n \geq n_0$“, tedy „pro všechna n až na konečně mnoho výjimek.“

Poznámka: $\mathcal{O}$-notace tedy vyjadřuje, že funkce f je skoro všude menší nebo nejvýše rovná nějakému reálnému násobku funkce g. Ačkoliv zápis vypadá jako rovnost, rozhodně není symetrický: například platí $\log n = \mathcal{O}(n)$, ale neplatí $n = \mathcal{O}(\log n)$. Formálně by bylo lepší považovat $\mathcal{O}(g)$ za třídu funkcí, pro které platí, že se dají shora odhadnout kladným násobkem funkce g, a psát tedy $f \in \mathcal{O}(g)$, ale zvyk je bohužel železná košile.

Příklady: $2{,}5n^2 = \mathcal{O}(n^2)$, $2{,}5n^2 + 30n = \mathcal{O}(n^2)$.

Také platí:

$$\mathcal{O}(f) + \mathcal{O}(g) \in \mathcal{O}(f + g),$$

čímž myslíme, že pokud vezmeme libovolnou $f' = O(f)$ a $g' = O(g)$, bude $f' + g' = O(f + g)$. To platí, jelikož skoro všude je $f' \leq cf$, $g' \leq dg$, a tedy $f' + g' \leq cf + dg \leq (c + d)(f + g)$.

Cvičení: Ukažte, že:

- $\mathcal{O}(f) \cdot \mathcal{O}(g) = \mathcal{O}(f \cdot g)$,
- $\mathcal{O}(f + g) = \mathcal{O}(\max(f, g))$,
- $\mathcal{O}(n^2) + \mathcal{O}(n) = \mathcal{O}(n^2 + n) = \mathcal{O}(n^2)$.

$\mathcal{O}$-notace popisuje horní odhad asymptotického chování funkce. Mnohdy však potřebujeme také odhadnout funkci zespodu (chceme-li říci, že algoritmus potřebuje *alespoň* nějaké množství času nebo paměti), případně z obou stran:

Definice:

- $f = \Omega(g) \equiv \exists c > 0 : \forall^* n \in \mathbb{N} : f(n) \geq c \cdot g(n)$.
  Ω-notace tedy říká, že hodnota funkce f je vždy stejná nebo vyšší než nějaký c-násobek funkce g, a tedy $g = \mathcal{O}(f)$.
- $f = \Theta(g) \equiv f = \mathcal{O}(g) \wedge f = \Omega(g)$
  nebo výřečněji:
  $f = \Theta(g) \equiv \exists\ c_1, c_2 > 0 : \forall^* n \in \mathbb{N} : c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n)$ To znamená, že existují nezáporné reálné konstanty c_1, c_2 takové, že se funkce $f(n)$ dá ohraničit c_1- a c_2-násobkem funkce $g(n)$.

Poznámka: Ona rovnost je dost zavádějící. Není to totiž rovnost, není to symetrické. Jiný, možná rozumnější pohled, je zavedení $\mathcal{O}(g)$ jako množiny. Pak by se dalo psát $f \in \mathcal{O}(g)$, nebo třeba $\mathcal{O}(n) \subseteq \mathcal{O}(n^2)$.

Porovnání růstu funkcí: (aneb jak moc máme algoritmy rádi podle jejich chování od nejlepších k nejhorším)

- $\Theta(1)\ldots$ funkce zespoda i shora ohraničené konstantami
- $\Theta(\log(\log n))$
- $\Theta(\log n)$ … logaritmická
- $\Theta(n^\varepsilon), \varepsilon \in (0, 1)$ … sublineární
- $\Theta(n)$ … lineární
- $\Theta(n^2)$ … kvadratická
- $\Theta(n^k)$ … polynomiální
- $\Theta(2^n)$ … exponenciální při základu 2
- $\Theta(3^n)$ … exponenciální při základu 3
- $\Theta(k^n)$ … exponenciální při základu $k > 1$
- $\Theta(n!)$ … faktoriálová
- $\Theta(n^n)$
- … nekonečně mnoho dalších tříd (i mezi těmi výše uvedenými)

Poznámka: Pokud se v odhadu složitosti vyskytne logaritmus (jinde než v exponentu), nezáleží na tom, jaký má základ, protože platí:

$$\log_k n = \frac{\log_c n}{\log_c k} = \frac{1}{\log_c k} \cdot \log_c n,$$

kde $1/\log_c k$ je jen konstanta, takže ji můžeme „schovat do $\mathcal{O}$“.
Příklad: Select sort (rozebraný výše): Když jej pustíme na n čísel, pak časová složitost je $T(n) = \Theta(n^2)$ a prostorová $S(n) = \Theta(n)$.

Úvod do grafových algoritmů

Další důležitou a zajímavou kapitolou jsou grafové algoritmy. Například následující příklady lze (i když to tak občas na první pohled nevypadá) řešit nějakým grafovým algoritmem:

- Mám mapku silniční sítě, v ní místa (vrcholy) označená „Doma“ a „Škola“. Dostanu se do školy (leží ve stejné komponentě souvislosti)? Dostanu se do školy, když v zimě napadne hodně sněhu a některé cesty budou neprůjezdné? A jaký nejkratší úsek cest musí silničáři prohrnout, aby byla všechna místa na mapě dostupná?
- Mějme hlavolam „Lloydova devítka“ – krabičku 3×3 se čtverečky označenými čísly od jedné do osmi a jednou mezerou, čtverečky jsou zamíchané a naším úkolem je správně je seřadit pomocí přesouvání čtverečků sousedících s mezerou do této mezery. Jak to udělat? Kolik nejméně kroků nám na to stačí? Jde to vůbec se zadáním, které jsme dostali?
- Jaké je nejkratší (kladné, celé) číslo v desítkové soustavě zapsané jen číslicemi 1, 0, které je dělitelné třinácti? Nakreslíme orientovaný graf s vrcholy 0 až 12 a hranami (x, y), $y = 10 \cdot x \bmod 13$ a $y = (10 \cdot x + 1) \bmod 13$ (z každého vrcholu vychází jedna hrana za přidání číslice 1 a další za číslici 0). Hledané číslo existuje právě tehdy, když graf obsahuje orientovaný sled z 1 do 0. Jakým algoritmem takový sled najdeme?

Podobné a další úlohy budeme řešit v následujících kapitolách.