Otázka 15 - Y36DSA

Zadání

Složitost algoritmů. Operační a paměťová složitost, operační složitost v průměrném, nejhorším a nejlepším případě, asymptotická složitost. (Y36DSA)

Slovníček pojmů

Složitost algoritmu určuje závislost *výpočetních prostředků* prováděného algoritmu na *velikosti vstupních dat*. Obecně může záviset nejen na velikosti vstupních dat, ale také na jejich hodnotách (pak říkáme, že algoritmus je citlivý na vstupní data).

Velikost vstupních dat závisí na specifikaci daného problému

- **počet bitu vstupního čísla,**
- **délka vstupní posloupnosti čísel,**
- **rozměry vstupní matice,**
- **počet znaků vstupního textu**
- **...**

Operační a paměťová složitost algoritmu jsou definovány jako *výpočetní prostředky algoritmu*. U operační složitosti se jedná o cykly procesoru, u paměťové složitosti se jedná o paměťové buňky.

Operační (časová, výpočetní) složitost se zabývá *dobou běhu algoritmu* a *jeho počtu operací* v závislosti na vstupní data.

Paměťová složitost udává celkové nároky algoritmu na operační paměť. Tzn. kolik si je třeba podržet informací / proměnných při běhu algoritmu.

Vyjádření složitosti algoritmu

- **Složitost algoritmu může obecně záviset na hodnotách vstupních dat.**
- **Proto je obecně potřebné vyjádřit složitost algoritmu:**
  - **v nejlepším případě,**
  - **v nejhorším případě,**
  - **v průměrném případě.**
- **Poslední případ je nejobtížnější, protože není vždy zřejmé, co to jsou vstupní data v průměrném případě. Typicky náhodně vygenerovaná, ale to nemusí být v praxi splněno.**
- **Toto vyjádření platí jak pro operační tak pro paměťovou složitost.**

Pro názornost si ukážeme na následujícím algoritmu všechny tři případy.

```
// algoritmus vracející index prvního výskytu hledaného čísla
// n udává počet prvků pole
// p je pole čísel, které prohledáváme
public int Najdi (int hledaneCislo)
{
   for (int i = 0; i < n; i++)
   {
       if(p[i] == hledaneCislo)
           return i;
   }
}
```

Nyní můžeme vyjádřit složitost algoritmu pro všechny případy:

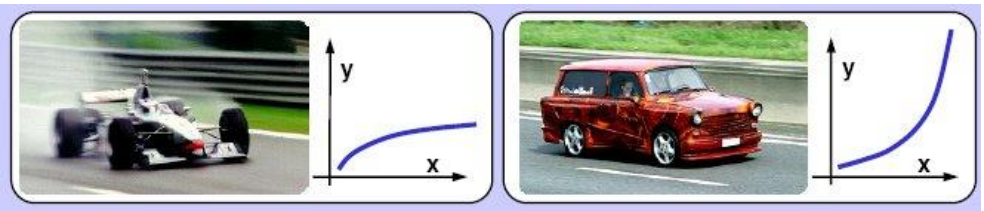
- ***Nejlepší případ* nastane, pokud hledané číslo bude hned na začátku prohledávaného pole. Provede se jen 1 iterace.**
- ***Nejhorší případ* nastane, pokud hledané číslo bude až na konci prohledávaného pole. To znamená, projet celé pole, což je *n* iterací.**
- ***Průměrný případ* nastane, pokud bude provedeno *n/2* iterací.**

(pozn. slovo *iterace* může být použito ve stejném významu jako opakování. V naší fci se iterací myslí jedno proběhnutí těla for cyklu)

Asymptotická složitost

- **U algoritmu nás především zajímá, jak se bude chovat pro velké ($\rightarrow \infty$) instance problému.**
- **Každému algoritmu lze přiřadit rostoucí funkci znázorňující závislost počtu provedených operací na vstupní data.**

Z následujících dvou obrázků je patrné, který algoritmus je rychlý (vlevo) a pomalý (vpravo). Osa X určuje množství vstupních dat a osa Y znázorňuje množství provedených operací.



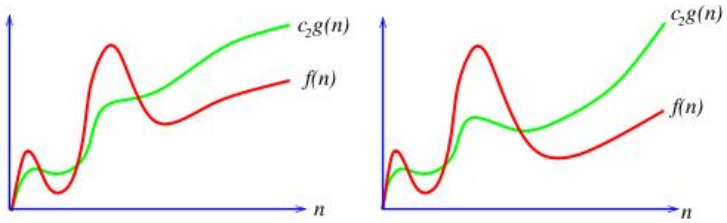
Pro následující definice budeme brát v úvahu následující:

- **ℵ+ je množina přirozených čísel**
- **ℜ+ je množina kladných reálných čísel**
- **Uvažujeme pouze funkce jedné kladné proměnné *n*:**
  - ***f, g,...:* ℵ+ → ℜ+.**

Asymptotická horní mez: O - notace (omikron)

Jsou - li dány funkce *f(n)* a *g(n)*, pak řekneme, že *f(n)* je nejvýše řádu jako *g(n)*. Psáno *f(n)* = *O(g(n))*, jestliže:

$\exists c_2 \in \Re+, \exists n_0 \in \aleph+; \forall n \geq n_0: f(n) \leq c_2 \cdot g(n)$.

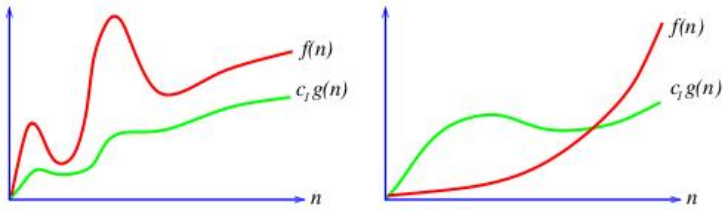


- **Zápisem *f(n)* = *O(g(n))* vyjadřujeme, že *g(n)* je pro *f(n)* asymptotickou horní mezí stejného nebo vyššího řádu.**
- **O - notaci používáme pro:**
  - **Vyjádření horní meze funkce až na multiplikativní konstantu.**
  - **Odhady složitostí algoritmů v nejhorších případech.**

Asymptotická dolní mez: Ω - notace (omega)

Jsou - li dány funkce *f(n)* a *g(n)*, pak řekneme, že *f(n)* je nejméně řádu *g(n)*. Psáno *f(n)* = *Ω(g(n))*, jestliže:

$\exists c_1 \in \Re+, \exists n_0 \in \aleph+; \forall n \geq n_0: c_1 \cdot g(n) \leq f(n)$.

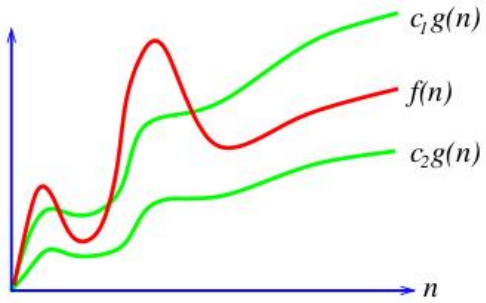


- **Zápisem *f(n)* = *Ω(g(n))* vyjadřujeme, že *g(n)* je pro *f(n)* asymptotickou dolní mezí stejného nebo nižšího řádu.**
- **Ω - notaci používáme pro:**
  - **Vyjádření dolní meze funkce až na multiplikativní konstantu.**
  - **Odhady složitostí algoritmů v nejlepších případech.**

Asymptotická těsná mez: Θ - notace (theta)

Jsou - li dány funkce *f(n)* a *g(n)*, pak řekneme, že *f(n)* je téhož řádu jako *g(n)*. Psáno *f(n)* = *Θ(g(n))*, jestliže:

$\exists c_1, c_2 \in \Re+, \exists n_0 \in \aleph+; \forall n \geq n_0: c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n)$.



Θ - notaci používáme pro vyjádření faktu, že 2 funkce jsou asymptoticky stejné až na multiplikativní konstantu.

Klasifikace algoritmů

Každému algoritmu můžeme přiřadit charakteristickou rostoucí funkci. Podle růstu funkce vyjadřujeme třídu chování Θ.

Konstantní složitost	**Θ(1)**
Logaritmická složitost	**Θ(log(n))**
Lineární složitost	**Θ(n)**
Kvadratická složitost	**$\Theta(n^2)$**
Kubická složitost	**$\Theta(n^3)$**
Polynomialní složitost	**$\Theta(n^k)$**
Exponenciální složitost	**$\Theta(k^n)$**

Porovnání funkcí podle jejich složitosti

$n^{1/\log n} < \log\log n < \sqrt{\log n} < \log n < \log^2 n < n \,/\, \log n < n < n \cdot \log n < n^2 \cdot \log n < n^3 < 2^{\log^2 n} < 2^{\sqrt{n}} < 1.5^n < 2^n < n2^n < e^n < n! < 2^{2\wedge n}$

Zákony asymptotické aritmetiky

Transitivita:	***$f(n) = O(g(n)) \wedge g(n) = O(h(n)) \Rightarrow f(n) = O(h(n))$***
	Podobně pro Ω, Θ
Reflexivita:	***$f(n) = O(f(n))$*. Podobně pro Ω, Θ**
Symetrie:	***$f(n) = \Theta(g(n)) \Leftrightarrow g(n) = \Theta(f(n))$***
Transpoziční symetrie:	***$f(n) = O(g(n)) \Leftrightarrow g(n) = \Omega(f(n))$***
Inkluze:	***$f(n) = \Theta(g(n)) \Rightarrow f(n) = O(g(n))$,***
	$f(n) = \Theta(g(n)) \Rightarrow f(n) = \Omega(g(n))$.

Pár příkladů pro lepší pochopení asymptotické složitosti:

10.
Pro rostoucí spojité fukce f(x), g(x) platí $f(x) \in \Omega(g(x))$. Z toho plyne, že

a) $f(x) \in O(g(x))$
b) $f(x) \in \Theta(g(x))$
c) $g(x) \in \Theta(f(x))$
d) $g(x) \in \Omega(f(x))$
e) $g(x) \in O(f(x))$

Řešení
$f(x) \in \Omega(g(x))$ znamená, že f(x) roste rychleji nebo stejně rychle jako g(x). Tudíž g(x) roste pomaleji nebo stejně rychle jako f(x). To vyjadřujeme zápisem $g(x) \in O(f(x))$. Platí možnost e).

11.
Pro rostoucí spojité fukce f(x), g(x) platí $f(x) \in O(g(x))$. Z toho plyne, že

a) $f(x) \in \Theta(g(x))$
b) $f(x) \in \Omega(g(x))$
c) $g(x) \in \Theta(f(x))$
d) $g(x) \in \Omega(f(x))$
e) $g(x) \in O(f(x))$

Řešení
$f(x) \in O(g(x))$ znamená, že f(x) roste pomaleji nebo stejně rychle jako g(x). Tudíž g(x) roste rychleji nebo stejně rychle jako f(x). To vyjadřujeme zápisem $g(x) \in \Omega(f(x))$. Platí možnost d).

12.
Pokud funkce f roste asymptoticky rychleji než funkce g (tj. $f(x) \notin O(g(x))$), platí následující tvrzení

a) jsou-li v bodě x definovány obě funkce, pak f(x) > g(x)
b) rozdíl f(x) - g(x) je vždy kladný
c) rozdíl f(x) - g(x) je kladný pro každé x > y, kde y je nějaké dostatečně velké číslo
d) obě funkce f i g jsou definovány jen pro nezáporné argumenty
e) nic z předchozího

Řešení
Řeč je o asymptotické složitosti, takže vztah funkcí f(x) a g(x) blízko nuly není nijak podstatný. Varianty a) b) a d) žádají z tohoto hlediska příliš mnoho a tedy neplatí. Tvrzení úlohy $f(x) \notin O(g(x))$ říká, že od určitého okamžiku musí platit f(x) > g(x), což je ekvivalentní variantě c).

13.
Pokud funkce f roste asymptoticky stejně rychle jako funkce g (tj. $f(x) \in \Theta(g(x))$), platí právě jedno následující tvrzení. Které?

a) jsou-li v bodě x definovány obě funkce, pak f(x) = g(x)
b) ani poměr f(x)/g(x) ani poměr g(x)/f(x) nekonverguje k nule s rostoucím x
c) rozdíl f(x) - g(x) je kladný pro každé x > y, kde y je nějaké dostatečně velké číslo

Třídy složitosti

Složitosti můžeme rozdělit do několika tříd složitosti. Nás boudou zajímat především třídy P a NP.

Třída složitosti P (polynomial)

P je obvykle považována za třídu problémů, které jsou efektivně řešitelné v polynomiálním čase. Jejich časová složitost je $O(n^k)$.

Třída složitosti NP (nedeterministicky polynomiální)

NP úlohy umíme prozatím řešit jen tak že řešení „uhádneme“ a ověříme. Není však znám žádný efektivní algoritmus řešení (avšak není vyloučeno, že existuje). Pro ověření „uhádnutého“ výsledku ovšem existuje algoritmus s polynomiální složitostí. To znamená, že pro získání jednoho výsledku potřebujeme v podstatě polynomiální čas. Pro získání všech řešení potřebujeme tedy součet těchto časů.

Nedeterministický algoritmus pro rozhodovací úlohy má obvykle dvě fáze:

- **nedeterministická fáze** - **do paměti se zapíše uhádnuté řešení**
- **deterministická fáze** - **použije se deterministický algoritmus pro určení, zda toto řešení představuje opravdu řešení zadané úlohy**

Pozn.: Víme že P patří do NP, ale to, zda P je podmnožina NP, nebo P = NP (tedy jestli pro řešení NP existuje nějaký efektivní polynomiálně složitý algoritmus), to je problém století.

Příklad:

Máme množinu čísel jako např. {-9, -6, -2, 5, 8} a chceme vědět jestli součet nějakých čísel z této množiny je nula. V tomto případě je odpovědí podmnožina {-2, -6, 8}. Vypadá to jednoduše, ovšem pokud zadáme velkou množinu čísel, náročnost problému rapidně vzroste. Žádný rozumný algoritmus zatím pro tento případ nebyl nalezen.

Na druhé straně, pokud z množiny vyjmeme zmíněnou podmnožinu, je snadné ověřit, jestli je její součet nula nebo ne. NP problémy mají společné to, že pro všechny existuje nějaký ověřovací algoritmus, který je časově polynomiálně složitý.

Problém B je NP-těžký, pokud pro libovolný problém A ze třídy NP platí, že A je polynomiálně redukovatelný na B.

Problém je NP-úplný, pokud patří do třídy NP a je NP-těžký.

Zdroje

- **Slajdy z přednášek předmětu DSA https://service.felk.cvut.cz/courses/X36DSA/ [https://service.felk.cvut.cz/courses/X36DSA/]**
- **Datové struktury a algoritmy z let 2004 - 2007 http://cmp.felk.cvut.cz/~berezovs/DSA/ [http://cmp.felk.cvut.cz/~berezovs/DSA/]**
- **Wikipedia http://cs.wikipedia.org/wiki/Asymptotick%C3%A1_slo%C5%BEitost [http://cs.wikipedia.org/wiki/Asymptotick%C3%A1_slo%C5%BEitost]**
- **MATFYZ - Obecná informatika, Algoritmy a datové struktury http://statnice.matfyz.info/generated/Obecna_informatika_3.pdf [http://statnice.matfyz.info/generated/Obecna_informatika_3.pdf]**

spolecne/spol15.txt · Poslední úprava: 2010/06/06 10:39 autor: Sway