Analýza programů

Z FAV wiki

Analýza programu se zabývá nároky programu na zdroje, tedy čas, paměť, šířka pásma... Máme-li problém, lze pro jeho řešení zpravidla použít více algoritmů, a každý algoritmus může mít několik implementací. Analýza programů se zabývá právě vztahem implementace řešení původního problému vzhledem k požadovanému zdroji.

Obsah

Časová náročnost

Nejčastěji nás zajímá časová náročnost programu. Analyzujeme, jakou dobu bude implementovanému algoritmu trvat výpočet. Tuto dobu ovlivňuje způsob implementace (počet nutně provedených operací) a velikost vstupních dat (počet prvků, které algoritmus zpracovává, případně jejich uspořádání). Definujeme elementární čas pro všechny operace $c_{operace}$ a jejich skádáním počítáme celkový čas potřebný pro provedení algoritmu.

Příklad: Naplnění již alokovaného pole náhodnými čísly (c#)

```
for (int i = 0, i < pole.Length; i++)
{
  pole[i] = rand.Next();
}
```

- máme pole délky n = pole.Length:
- alokace proměnné trvá c_{aloc}
- operace přiřazení trvá $c_{přiřazení}$
- operace porovnání trvá $c_{porovnání}$
- operaci inkrementace lze definovat jako $c_{inkrement} = c_{přiřazení} + c_{součet}$
- řekněme, že generování náhodného čísla trvá konstantní čas c_{rand}

- Inicializace cyklu for: alokace + přiřazení = $c_{aloc} + c_{přiřazení}$
- Běh cyklu: porovnání (i < length) + přiřazení (do pole) + náh. číslo + inkrementace (i++) = $c_{porovnání} + c_{přiřazení} + c_{rand} + c_{inkrement}$

- Cyklus for proběkne n-krát, tedy: n * ($c_{porovnání} + c_{přiřazení} + c_{rand} + c_{inkrement}$)
- Celkem: $c_{aloc} + c_{přiřazení} + n * (c_{porovnání} + c_{přiřazení} + c_{rand} + c_{inkrement})$

Asymptotická složitost

Ve většině případů neznáme čas potřebný pro konstantní operace, ale víme, že je velmi malý. Dobu výpočtu předchozího příkladu můžeme tedy přepsat na **a** + **n** * **b**, kde **a** a **b** jsou doby, které mají určitou délku, která nás však přísliš nezajímá.

Více než doba výpočtu nás tedy zajímá počet operací, které je třeba vykonat (stejně budou operace na každém systému vykonávány jinou dobu (PC vs. mobilní telefon např.)). Složitostí programu myslíme právě počet operací potřebný pro výpočet vzhledem k počtu prvků na vstupu algoritmu, omezený shora a/nebo zdola. Tato složitost se pro libovolný počet prvků asymptoticky blíží k určité funkci. Nás zajímá, ke které. Rozlišujeme také složitost algoritmu a složitost problému (ta by měla patřit do otázky PPA1).

Theta notace Θ(f(x)) vyjadřuje A.S., omezenou složitostní funkcí $c_{1,2} * f(x)$ shora, resp zdola. Hledáme tedy 2 konstatnty, pro které je analyzovaná složitost omezena.

Tato notace říká, že algoritmus nebude asymptoticky složitější než $c_1 * f(x)$, a nebude rychlejší než $c_2 * f(x)$. To znamená, že problém lze řešit algoritmem, který nebude asymptoticky složitější než $c_1 * f(x)$, ale zároveň nikdy nebude lepší než $c_2 * f(x)$.

Omikron notace O(f(x)) vyjadřuje A.S. omezenou funkcí c * f(x) pouze shora. Jinými slovy, jde o maximální možnou složitost algoritmu. Jde o první podmínku theta notace.

Omega notace Ω(f(x)) vyjadřuje A.S. omezenou funkcí c * f(x) pouze zdola. Říká, že algoritmus pro alespoň jeden vstup bude této složitosti. Druhá podmínka theta notace.

Pokud platí omikron a omega notace pro stejnou f(x), mluvíme o theta notaci.

Konstanty v notaci zanedbáváme, jelikož vzhledem k většímu n nehrají roli. Sčítáním složitostí tedy nedochází k jejich zhoršení, jsou-li stejného řádu (např $n^2 + n^2 = 2n^2 => n^2$, 200 operací je asymptoticky stejné jako 100, ale $n^2 + n^3 => n^3$, 100 * 100 operací je vzhledem k 100 * 100 * 100 zanedbatelné).

Předchozí příklad je tedy složitosti Θ(n) , jelikož je vždy lineární vzhledem k délce pole (netrvá kratší, ani delsí dobu/počet operací)

Očekávaná složitost

Většina algoritmů je asymptoticky omezena pro extrémní případny na vstupu (např. Quicksort je Ω(n) (pole již seřazeno) a O(n2) (pole je seřazeno opačně)). Chování algoritmu má však očekávanou složitost, která nastává pro většinu případů (jak víme O(n logn) pro QuickSort). Využívá se také Theta/Omikron/Omega notací.

Je-li algoritmus závislý na vstupních datech (resp. na jejich uspořádání), a potřebujeme, aby složitost byla pokud možno vždy očekávaná, lze využít náhodného přeuspořádání dat na vstupu (za předpokladu, že je to možné - např. pro řazení ano, pro výpis znaků na obrazovku evidentně ne). Toto přeuspořádání může být konstatní

složitosti (např 100x přeházíme vzájemně prvky pole s náhodnými indexy pokud je pole rozumné odpovídající délky), případně lineární, složitost algoritmu tedy neovlivňuje (za předpokladu, že algoritmus samotný není očekávané sublineární časové složitosti).

Klasifikace složitosí

Polynomiální - složitost O(n * c) kde c je konstanta. Polynomiální algoritmy jsou použitelné. Ideální jsou algoritmy rychlejší (lineární, logaritmické,...), ne vždy je však možné takový algoritmus vymyslet.

Exponenciální - složitost $O(c^n)$. Tyto algoritmy nejsou příliš dobré, a je vhodné se zamyslet a pokusit se vymyslet jiný, polynomiálně složitý algoritmus. Zpravidla algoritmy hrubé síly jsou této složitosti, jelikož procházejí všechny kombinace vstupu a hledají řešení. Pro malé n jsou však použitelné díky menší režii a přípravě, paměťovým nárokům atd.

Postupy výše lze aplikovat i na složitosti paměťové, šířky pásma, ...