Asymptotická složitost

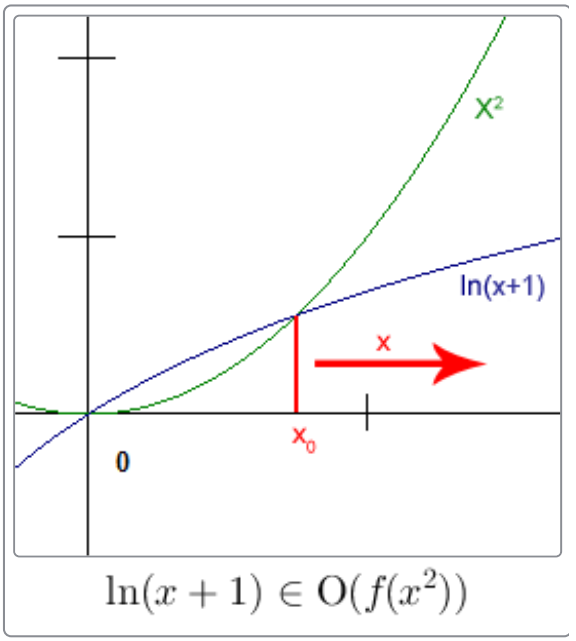
Složitost algoritmu udává, jak je daný algoritmus rychlý (kolik provede elementárních operací) vzhledem k množině vstupních dat. Ke klasifikaci algoritmů se obvykle používá tzv. asymtotická složitost, což je rozdělení algoritmů do tříd složitostí, u kterých platí, že od určité velikosti dat, je algoritmus dané třídy vždy pomalejší než algoritmus třídy předchozí, bez ohledu na to, jestli je některý z počítačů *c*-násobně výkonnější (*c* je konstanta).

Škála v nekonečnu slouží k rozlišení jednotlivých tříd. Říká, že pokud se *n* blíží k nekonečnu, tak neexistuje reálná konstanta taková, aby byl algoritmus z vyšší třídy rychlejší než ten z třídy přechozí.

$$1 \ll \log(n) \ll n \ll n \cdot \log(n) \ll n^k \ll k^n \ll n! \ll n^n$$

Co nám to vlastně říká?

Pokud máme dva algoritmy o srovnatelné složitosti, první $O(n)$ a druhý $O(2n)$, tak nám stačí ten druhý pustit na 2x rychlejším stroji a nepoznáme rozdíl. Pokud ovšem nejsou ve stejné třídě složitosti, například jeden $O(n)$ a druhý $O(n^2)$, tak nám na srovnání výkonu nepomůže libovolně výkonný počítač, protože dvojnásobný objem dat bude druhému algoritmu trvat 4x tolik času, desetinásobný 100x tolik času.



Jednoduše řečeno: pokud spadají dva algoritmy do různých tříd asymptotické složitosti, pak vždy existuje takové množství dat, od kterého je asymptoticky lepší algoritmus vždy rychlejší, bez ohledu na to, kolikrát je některý z počítačů výkonnější.

Na obrázku vidíme, že i kdybychom přenásobili libovolnou, byť velmi malou, konstantou *c* funkci x^2 (tj. zrychlovali bychom počítač, na němž tento algoritmus běží), tak vždy bude existovat bod x_0, od kterého bude algoritmus popsaný logaritmickou funkcí rychlejší na všech datech velikosti $x > x_0$. Změnou konstanty *c* bychom pouze posouvali bod x_0 po ose *x*.

Jak se to počítá

Složitost můžeme spočítat několika způsoby - jako počet elementárních operací, případně jednodušeji jako počet elementárních operací nad daty, nebo dokonce pouze jako počet porovnání. Všechny tyto metody dávají obvykle stejný výsledek.

Řád růstu funkcí

U většiny algoritmů nelze říci, že jejich složitost odpovídá přesně jedné třídě, protože rychlost algoritmu závisí také na povaze dat. Z tohoto důvodu se používáme řád růstu funkcí, který zohledňuje nejhorší i nejlepší možný běh algoritmu. O algoritmu tak například řekneme, že je v $\Omega(n)$ a $O(n^2)$, což znamená, že nikdy nedoběhne rychleji než v lineárním čase, ale na druhou stranu jeho složitost není pro žádná data horší než kvadratická.

$O(f(x))$ - Omicron(f(x)) – algoritmus probíhá asymptoticky stejně rychle nebo rychleji než $f(x)$.

$$O(f(x)) = \{g(x);\ \exists x_0 > 0,\ c > 0,\ \forall x > x_0 : g(x) < c \cdot f(x)\}$$

$\Omega(f(x))$ – Omega(f(x)) – algoritmus probíhá asymptoticky stejně rychle nebo pomaleji než $f(x)$.

$$\Omega(f(x)) = \{g(x);\ \exists x_0 > 0,\ c > 0,\ \forall x > x_0 : c \cdot f(x) < g(x)\}$$

$\Theta(f(x))$ – Theta(f(x)) – algoritmus probíhá asymptoticky stejně rychle jako $f(x)$. Tj. je zároveň v $O(f(x))$ a v $\Omega(f(x))$.

$$\Theta(f(x)) = \{g(x);\ \exists x_0 > 0,\ c_1 > 0,\ c_2 > 0,\ \forall x > x_0 : c_1 \cdot f(x) < g(x) < c_2 \cdot f(x)\}$$

Příklad

Java

```java
/**
 * Bublinkove razeni
 * @param array pole k serazeni
 */
public static void bubbleSort(int[] array){
    for (int i = 0; i < array.length - 1; i++) {
        for (int j = 0; j < array.length - i - 1; j++) {
            if(array[j] < array[j+1]){
                int tmp = array[j];
                array[j] = array[j+1];
                array[j+1] = tmp;
            }
        }
    }
}
```

Vnitřní cyklus tohoto algoritmu - *bubble sortu* - proběhne $(n-1) + (n-2) + (n-3) + ... + 1$ krát, což je podle součtu aritmetické posloupnosti $n \cdot ((n-1)+1)/2 = (n^2/2)$ operací. Protože počítáme asymptotickou složitost, která je definována tak, že na multiplikativních konstantách nijak nezáleží, tak je můžeme vyškrtnout. Ze stejného důvodu bychom vyškrtli i případné aditivní konstanty (aditivní konstantu lze vždy přebít pomocí multiplikativní). Tímto očištěním získáme výslednou složitost n^2. Protože se v tomto případě jedná jak o nejhorší, tak o nejlepší případ běhu algoritmu, tak je celková asymptotická složitost $\Theta(n^2)$.

– Pavel Mička
Zakladatel a administrátor stránek Algoritmy.net