

Složitost algoritmů

- Stejný problém může řešit více různých algoritmů
- Potřebovali bychom rozhodnout, který algoritmus je lepší
- Nestačí algoritmy naprogramovat a změřit čas výpočtu
- Potřebovali bychom nějakou kvantitativní charakteristiku (složitost), podle které budeme algoritmy srovnávat
- Nejvhodnější je složitost definovat jako funkci velikosti vstupu

Problém „Vyhledávání“

Vstup: Sekvence prvků x, a_1, a_2, \dots, a_n .

Výstup: Pokud $a_i = x$ je výstupem i (pokud jich je více, tak nejmenší), jinak vypíše 0.

```

                                READ   R3
                                LOAD   $0,R1
cyklus:                        READ   R2
                                JZERO  R2,nenasel
                                ADD    $1,R1
                                SUB    R3,R2
                                JZERO  R2,nasel
                                JUMP   cyklus
nenasel:                        LOAD   $0,R1
nasel:                          WRITE R1
                                END
```

| | | |
|----------|-------|------------|
| | READ | R3 |
| | LOAD | \$0,R1 |
| cyklus: | READ | R2 |
| | JZERO | R2,nenasel |
| | ADD | \$1,R1 |
| | SUB | R3,R2 |
| | JZERO | R2,nasel |
| | JUMP | cyklus |
| nenasel: | LOAD | \$0,R1 |
| nasel: | WRITE | R1 |
| | END | |

Pro sekvenci

9, 1, 8, 9, 7, 2, 0

R1

R2

R3

Výstup:

Instrukcí: 0

| | | |
|----------|-------|------------|
| | READ | R3 |
| | LOAD | \$0,R1 |
| cyklus: | READ | R2 |
| | JZERO | R2,nenasel |
| | ADD | \$1,R1 |
| | SUB | R3,R2 |
| | JZERO | R2,nasel |
| | JUMP | cyklus |
| nenasel: | LOAD | \$0,R1 |
| nasel: | WRITE | R1 |
| | END | |

Pro sekvenci
9, 1, 8, 9, 7, 2, 0

R1
R2
R3 9

Výstup:

Instrukcí: 1

| | | |
|----------|-------|------------|
| | READ | R3 |
| | LOAD | \$0,R1 |
| cyklus: | READ | R2 |
| | JZERO | R2,nenasel |
| | ADD | \$1,R1 |
| | SUB | R3,R2 |
| | JZERO | R2,nasel |
| | JUMP | cyklus |
| nenasel: | LOAD | \$0,R1 |
| nasel: | WRITE | R1 |
| | END | |

Pro sekvenci
9, 1, 8, 9, 7, 2, 0

R1 0
R2
R3 9

Výstup:

Instrukcí: 2

| | | |
|----------|--------------|------------|
| | READ | R3 |
| | LOAD | \$0,R1 |
| cyklus: | READ | R2 |
| | JZERO | R2,nenasel |
| | ADD | \$1,R1 |
| | SUB | R3,R2 |
| | JZERO | R2,nasel |
| | JUMP | cyklus |
| nenasel: | LOAD | \$0,R1 |
| nasel: | WRITE | R1 |
| | END | |

Pro sekvenci
9, 1, 8, 9, 7, 2, 0

| | |
|----|---|
| R1 | 0 |
| R2 | 1 |
| R3 | 9 |

Výstup:

Instrukcí: 3

| | | |
|----------|-------|------------|
| | READ | R3 |
| | LOAD | \$0,R1 |
| cyklus: | READ | R2 |
| | JZERO | R2,nenasel |
| | ADD | \$1,R1 |
| | SUB | R3,R2 |
| | JZERO | R2,nasel |
| | JUMP | cyklus |
| nenasel: | LOAD | \$0,R1 |
| nasel: | WRITE | R1 |
| | END | |

Pro sekvenci
9, 1, 8, 9, 7, 2, 0

| | |
|----|---|
| R1 | 0 |
| R2 | 1 |
| R3 | 9 |

Výstup:

Instrukcí: 4

| | | |
|----------|------------|------------|
| | READ | R3 |
| | LOAD | \$0,R1 |
| cyklus: | READ | R2 |
| | JZERO | R2,nenasel |
| | ADD | \$1,R1 |
| | SUB | R3,R2 |
| | JZERO | R2,nasel |
| | JUMP | cyklus |
| nenasel: | LOAD | \$0,R1 |
| nasel: | WRITE | R1 |
| | END | |

Pro sekvenci

9, 1, 8, 9, 7, 2, 0

R1 1

R2 1

R3 9

Výstup:

Instrukcí: 5

Časová složitost

| | | |
|----------|--------------|------------|
| | READ | R3 |
| | LOAD | \$0,R1 |
| cyklus: | READ | R2 |
| | JZERO | R2,nenasel |
| | ADD | \$1,R1 |
| | SUB | R3,R2 |
| | JZERO | R2,nasel |
| | JUMP | cyklus |
| nenasel: | LOAD | \$0,R1 |
| nasel: | WRITE | R1 |
| | END | |

Pro sekvenci

9, 1, 8, 9, 7, 2, 0

| | |
|----|----|
| R1 | 1 |
| R2 | -8 |
| R3 | 9 |

Výstup:

Instrukcí: 6

| | | |
|----------|-------|------------|
| | READ | R3 |
| | LOAD | \$0,R1 |
| cyklus: | READ | R2 |
| | JZERO | R2,nenasel |
| | ADD | \$1,R1 |
| | SUB | R3,R2 |
| | JZERO | R2,nasel |
| | JUMP | cyklus |
| nenasel: | LOAD | \$0,R1 |
| nasel: | WRITE | R1 |
| | END | |

Pro sekvenci

9, 1, 8, 9, 7, 2, 0

| | |
|----|----|
| R1 | 1 |
| R2 | -8 |
| R3 | 9 |

Výstup:

Instrukcí: 7

| | | |
|----------|-------|------------|
| | READ | R3 |
| | LOAD | \$0,R1 |
| cyklus: | READ | R2 |
| | JZERO | R2,nenasel |
| | ADD | \$1,R1 |
| | SUB | R3,R2 |
| | JZERO | R2,nasel |
| | JUMP | cyklus |
| nenasel: | LOAD | \$0,R1 |
| nasel: | WRITE | R1 |
| | END | |

Pro sekvenci
9, 1, 8, 9, 7, 2, 0

| | |
|----|----|
| R1 | 1 |
| R2 | -8 |
| R3 | 9 |

Výstup:

Instrukcí: 8

| | | |
|----------|-------|------------|
| | READ | R3 |
| | LOAD | \$0,R1 |
| cyklus: | READ | R2 |
| | JZERO | R2,nenasel |
| | ADD | \$1,R1 |
| | SUB | R3,R2 |
| | JZERO | R2,nasel |
| | JUMP | cyklus |
| nenasel: | LOAD | \$0,R1 |
| nasel: | WRITE | R1 |
| | END | |

Pro sekvenci
9, 1, 8, 9, 7, 2, 0

| | |
|----|---|
| R1 | 1 |
| R2 | 8 |
| R3 | 9 |

Výstup:

Instrukcí: 9

```

      READ   R3
      LOAD   $0,R1
cyklus: READ   R2
      JZERO  R2,nenasel
      ADD    $1,R1
      SUB    R3,R2
      JZERO  R2,nasel
      JUMP   cyklus
nenasel: LOAD   $0,R1
nasel:  WRITE  R1
      END
```

Pro sekvenci
9, 1, 8, 9, 7, 2, 0

R1 1
R2 8
R3 9

Výstup:

Instrukcí: 10

| | | |
|----------|------------|------------|
| | READ | R3 |
| | LOAD | \$0,R1 |
| cyklus: | READ | R2 |
| | JZERO | R2,nenasel |
| | ADD | \$1,R1 |
| | SUB | R3,R2 |
| | JZERO | R2,nasel |
| | JUMP | cyklus |
| nenasel: | LOAD | \$0,R1 |
| nasel: | WRITE | R1 |
| | END | |

Pro sekvenci

9, 1, 8, 9, 7, 2, 0

R1 2

R2 8

R3 9

Výstup:

Instrukcí: 11

| | | |
|----------|--------------|------------|
| | READ | R3 |
| | LOAD | \$0,R1 |
| cyklus: | READ | R2 |
| | JZERO | R2,nenasel |
| | ADD | \$1,R1 |
| | SUB | R3,R2 |
| | JZERO | R2,nasel |
| | JUMP | cyklus |
| nenasel: | LOAD | \$0,R1 |
| nasel: | WRITE | R1 |
| | END | |

Pro sekvenci
9, 1, 8, **9**, **7**, **2**, **0**

| | |
|----|----|
| R1 | 2 |
| R2 | -1 |
| R3 | 9 |

Výstup:

Instrukcí: 12

| | | |
|----------|-------------|------------|
| | READ | R3 |
| | LOAD | \$0,R1 |
| cyklus: | READ | R2 |
| | JZERO | R2,nenasel |
| | ADD | \$1,R1 |
| | SUB | R3,R2 |
| | JZERO | R2,nasel |
| | JUMP | cyklus |
| nenasel: | LOAD | \$0,R1 |
| nasel: | WRITE | R1 |
| | END | |

Pro sekvenci

9, 1, 8, 9, 7, 2, 0

R1 2

R2 -1

R3 9

Výstup:

Instrukcí: 13

| | | |
|----------|-------|------------|
| | READ | R3 |
| | LOAD | \$0,R1 |
| cyklus: | READ | R2 |
| | JZERO | R2,nenasel |
| | ADD | \$1,R1 |
| | SUB | R3,R2 |
| | JZERO | R2,nasel |
| | JUMP | cyklus |
| nenasel: | LOAD | \$0,R1 |
| nasel: | WRITE | R1 |
| | END | |

Pro sekvenci
9, 1, 8, 9, 7, 2, 0

| | |
|----|----|
| R1 | 2 |
| R2 | -1 |
| R3 | 9 |

Výstup:

Instrukcí: 14

| | | |
|----------|-------|------------|
| | READ | R3 |
| | LOAD | \$0,R1 |
| cyklus: | READ | R2 |
| | JZERO | R2,nenasel |
| | ADD | \$1,R1 |
| | SUB | R3,R2 |
| | JZERO | R2,nasel |
| | JUMP | cyklus |
| nenasel: | LOAD | \$0,R1 |
| nasel: | WRITE | R1 |
| | END | |

Pro sekvenci
9, 1, 8, 9, 7, 2, 0

| | |
|----|---|
| R1 | 2 |
| R2 | 9 |
| R3 | 9 |

Výstup:

Instrukcí: 15

| | | |
|----------|-------|------------|
| | READ | R3 |
| | LOAD | \$0,R1 |
| cyklus: | READ | R2 |
| | JZERO | R2,nenasel |
| | ADD | \$1,R1 |
| | SUB | R3,R2 |
| | JZERO | R2,nasel |
| | JUMP | cyklus |
| nenasel: | LOAD | \$0,R1 |
| nasel: | WRITE | R1 |
| | END | |

Pro sekvenci

9, 1, 8, 9, 7, 2, 0

R1 2

R2 9

R3 9

Výstup:

Instrukcí: 16

| | | |
|----------|------------|------------|
| | READ | R3 |
| | LOAD | \$0,R1 |
| cyklus: | READ | R2 |
| | JZERO | R2,nenasel |
| | ADD | \$1,R1 |
| | SUB | R3,R2 |
| | JZERO | R2,nasel |
| | JUMP | cyklus |
| nenasel: | LOAD | \$0,R1 |
| nasel: | WRITE | R1 |
| | END | |

Pro sekvenci
9, 1, 8, 9, **7, 2, 0**

| | |
|----|---|
| R1 | 3 |
| R2 | 9 |
| R3 | 9 |

Výstup:

Instrukcí: 17

| | | |
|----------|--------------|------------|
| | READ | R3 |
| | LOAD | \$0,R1 |
| cyklus: | READ | R2 |
| | JZERO | R2,nenasel |
| | ADD | \$1,R1 |
| | SUB | R3,R2 |
| | JZERO | R2,nasel |
| | JUMP | cyklus |
| nenasel: | LOAD | \$0,R1 |
| nasel: | WRITE | R1 |
| | END | |

Pro sekvenci

9, 1, 8, 9, **7, 2, 0**

R1 3

R2 0

R3 9

Výstup:

Instrukcí: 18

```

cyklus:  READ   R3
         LOAD   $0,R1
         READ   R2
         JZERO  R2,nenasel
         ADD    $1,R1
         SUB    R3,R2
         JZERO  R2,nasel
         JUMP   cyklus
nenasel: LOAD   $0,R1
nasel:   WRITE  R1
         END
```

Pro sekvenci
9, 1, 8, 9, 7, 2, 0

R1 3
R2 0
R3 9

Výstup:

Instrukcí: 19

```
cyklus:  READ  R3
         LOAD  $0,R1
         READ  R2
         JZERO R2,nenasel
         ADD   $1,R1
         SUB   R3,R2
         JZERO R2,nasel
nenasel: JUMP   cyklus
nasel:   LOAD  $0,R1
nasel:   WRITE R1
        END
```

Pro sekvenci
9, 1, 8, 9, 7, 2, 0

R1 3
R2 0
R3 9

Výstup: 3

Instrukcí: 20

| | | | |
|----------|-------|-------------|---------------------|
| | READ | R3 | |
| | LOAD | \$0,R1 | Pro sekvenci |
| cyklus: | READ | R2 | 9, 1, 8, 9, 7, 2, 0 |
| | JZERO | R2, nenasel | |
| | ADD | \$1,R1 | R1 3 |
| | SUB | R3,R2 | R2 0 |
| | JZERO | R2, nasel | R3 9 |
| | JUMP | cyklus | |
| nenasel: | LOAD | \$0,R1 | Výstup: 3 |
| nasel: | WRITE | R1 | |
| | END | | Instrukcí: 21 |

- Počet instrukcí v uvedeném příkladě: $2 + 2 * 6 + 5 + 2 = 21$
- Pokud najde i , tž. $a_i = x$: $2 + (i - 1) * 6 + 5 + 2 = 6i + 3$
- Pokud nenajde takové i : $2 + n * 6 + 2 + 3 = 6n + 7$

Definice

Délka výpočtu algoritmu \mathcal{A} na vstupu x rozumíme počet kroků stroje RAM implementujícího algoritmus \mathcal{A} , které vykoná na vstupu x až do svého zastavení.

Definice

Časová složitost algoritmu \mathcal{A} (v nejhorším případě) je funkce $t_{\mathcal{A}} : \mathbb{N} \rightarrow \mathbb{N}$, kde $t_{\mathcal{A}}(n)$ udává maximální délku výpočtu mezi všemi vstupy x délky n .

Poznámka: Předpokládá se, že výpočet stroje RAM vždy skončí. Jako velikost vstupu uvažujeme počet znaků, kterými je vstup zapsán.

Příklad: V příkladu s vyhledáváním uvažujme, že na vstupu jsou jen jednobytová čísla. Abeceda k zapsání vstupu je $\{0, 1, \dots, 255\}$. Nejhorší případ nastává, pokud není nalezeno hledané číslo.

Složitost je $t_{\mathcal{A}}(n) = (n - 2) * 6 + 7 = 6n - 5$.

Definice

Časová složitost algoritmu \mathcal{A} **v nejlepším případě** je funkce $t_{\mathcal{A}} : \mathbb{N} \rightarrow \mathbb{N}$, kde $t_{\mathcal{A}}(n)$ udává minimální délku výpočtu mezi všemi vstupy x délky n .

Příklad: V našem případě je časová složitost v nejlepším případě konstantní funkce $t_{\mathcal{A}}(n) = 9$. Nejrychleji totiž výpočet skončí, jestliže hned první číslo prohledávané posloupnosti je ono hledané.

Poznámka: Časovou složitost v nejlepším případě většinou nemá smysl uvažovat. Jde často o konstantní funkci nebo je dána nějakými triviálními vstupy.

Definice

Časová složitost algoritmu \mathcal{A} v **průměrném případě** je funkce $t_{\mathcal{A}} : \mathbb{N} \rightarrow \mathbb{N}$, kde $t_{\mathcal{A}}(n)$ udává průměrnou délku výpočtu mezi všemi vstupy x délky n (neboli střední hodnotu délky výpočtu na pravděpodobnostním prostoru všech vstupů).

Příklad: Uvažujme v našem vyhledávání jednobytová čísla. Spočítáme průměrnou složitost pro $n = 5$ (uvažujeme jen správné vstupy délky 5).

- $x,x,y,z,0$: možností: 255^3 , délka výpočtu: $6 \cdot 1 + 3 = 9$
- $x,y,x,z,0$: možností: $255^2 \cdot 254$, délka výpočtu: $6 \cdot 2 + 3 = 15$
- $x,y,z,x,0$: možností: $255 \cdot 254^2$, délka výpočtu: $6 \cdot 3 + 3 = 21$
- $x,y,z,u,0$: možností: $255^4 - (255^3 + 255^2 \cdot 254 + 255 \cdot 254^2)$, délka výpočtu: $6 \cdot 3 + 7 = 25$
- Průměr tedy je $t_{\mathcal{A}}(5) = (255^3 \cdot 9 + 255^2 \cdot 254 \cdot 15 + 255 \cdot 254^2 \cdot 21 + (255^4 - (255^3 + 255^2 \cdot 254 + 255 \cdot 254^2)) \cdot 25) / 255^4 = 24.88$

Obecně by se dala složitost vyjádřit:

$$t_{\mathcal{A}}(n) = \frac{(255^{n-1} - \sum_{i=1}^{n-2} 255^i 254^{n-2-i})(6n - 5)}{255^{n-1}} + \\ + \frac{\sum_{i=1}^{n-2} (255^i 254^{n-2-i} (6n - 6i - 3))}{255^{n-1}}$$

Poznámka:

- Časová složitost v průměrném případě většinou nejvíce vypovídá o reálné složitosti algoritmu v praxi.
- Je obvykle nejtěžší ji vyjádřit.
- Často není podstatný rozdíl mezi složitostí v průměrném a nejhorším případě. Někdy je ale rozdíl významný.

Rychlost růstu funkcí

Program zpracovává vstup velikosti n .

Předpokládejme, že pro vstup velikosti n provede $f(n)$ operací, a že provedení jedné operace trvá $1 \mu\text{s}$ (10^{-6} s).

| | n | | | | | | | |
|------------|------------------|--------------------------|--------------------------|---------------------------|---------------------------|------------------------|---------------------------|-----------|
| $f(n)$ | 20 | 40 | 60 | 80 | 100 | 200 | 500 | 1000 |
| n | $20 \mu\text{s}$ | $40 \mu\text{s}$ | $60 \mu\text{s}$ | $80 \mu\text{s}$ | 0.1 ms | 0.2 ms | 0.5 ms | 1 ms |
| $n \log n$ | $86 \mu\text{s}$ | 0.213 ms | 0.354 ms | 0.506 ms | 0.664 ms | 1.528 ms | 4.48 ms | 9.96 ms |
| n^2 | 0.4 ms | 1.6 ms | 3.6 ms | 6.4 ms | 10 ms | 40 ms | 0.25 s | 1 s |
| n^3 | 8 ms | 64 ms | 0.216 s | 0.512 s | 1 s | 8 s | 125 s | 16.7 min. |
| n^4 | 0.16 s | 2.56 s | 12.96 s | 42 s | 100 s | 26.6 min. | 17.36 hod. | 11.57 dní |
| 2^n | 1.05 s | 12.75 dní | 36560 let | $38.3 \cdot 10^9$ let | $40.1 \cdot 10^{15}$ let | $50 \cdot 10^{45}$ let | $10.4 \cdot 10^{136}$ let | - |
| $n!$ | 77147 let | $2.59 \cdot 10^{34}$ let | $2.64 \cdot 10^{68}$ let | $2.27 \cdot 10^{105}$ let | $2.96 \cdot 10^{144}$ let | - | - | - |

Rychlost růstu funkcí

Uvažujme 3 algoritmy se složitostmi $t_1(n) = n$, $t_2(n) = n^3$, $t_3(n) = 2^n$. Náš počítač zvládne v reálném čase (kolik jsme ochotni počkat) 10^{12} kroků.

| Složitost | Velikost vstupu |
|----------------|-----------------|
| $t_1(n) = n$ | 10^{12} |
| $t_2(n) = n^3$ | 10^4 |
| $t_3(n) = 2^n$ | 40 |

Rychlost růstu funkcí

Uvažujme 3 algoritmy se složitostmi $t_1(n) = n$, $t_2(n) = n^3$, $t_3(n) = 2^n$. Náš počítač zvládne v reálném čase (kolik jsme ochotni počkat) 10^{12} kroků.

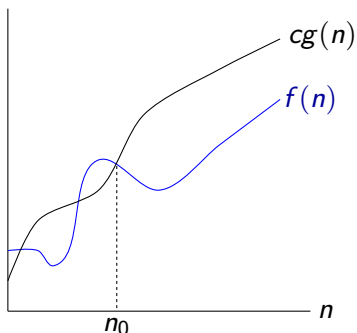
| Složitost | Velikost vstupu |
|----------------|-----------------|
| $t_1(n) = n$ | 10^{12} |
| $t_2(n) = n^3$ | 10^4 |
| $t_3(n) = 2^n$ | 40 |

Nyní počítač 1000 násobně zrychlíme. Zvládne tedy 10^{15} kroků.

| Složitost | Velikost vstupu | Nárůst |
|----------------|-----------------|--------------|
| $t_1(n) = n$ | 10^{15} | $1000\times$ |
| $t_2(n) = n^3$ | 10^5 | $10\times$ |
| $t_3(n) = 2^n$ | 50 | +10 |

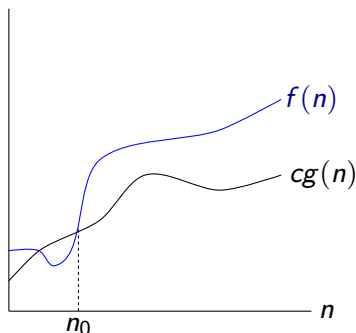
- Přesnou složitost bývá problém vyjádřit
- Přesná složitost je silně závislá na konkrétním zvoleném modelu a konkrétní implementaci
- Složitost nás většinou zajímá hlavně pro velké vstupy. Pro malé vstupy obvykle i neefektivní algoritmus proběhne rychle
- Ve většině případů pro představu o složitosti stačí nejvýznamnější člen složitostní funkce
- Proto zavádíme tzv. asymptotickou notaci, která nám umožní zanedbat konstanty a méně významné členy funkcí

Asymptotická notace



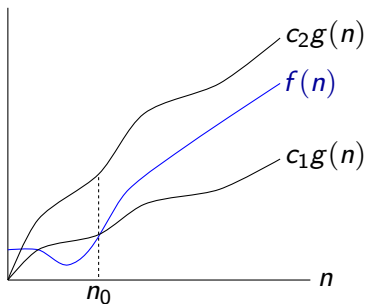
Definice

Pro libovolné dvě funkce $f, g : \mathbb{N} \rightarrow \mathbb{N}$ řekneme, že $f \in O(g)$, označujeme také jako $f(n) \in O(g(n))$, právě tehdy, když platí $(\exists c > 0)(\exists n_0 \geq 0)(\forall n \geq n_0) : f(n) \leq c g(n)$.



Definice

Pro libovolné dvě funkce $f, g : \mathbb{N} \rightarrow \mathbb{N}$ řekneme, že $f \in \Omega(g)$, označujeme také jako $f(n) \in \Omega(g(n))$, právě tehdy, když platí $g \in O(f)$ neboli $(\exists c > 0)(\exists n_0 \geq 0)(\forall n \geq n_0) : 0 \leq c g(n) \leq f(n)$.



Definice

Pro libovolné dvě funkce $f, g : \mathbb{N} \rightarrow \mathbb{N}$ řekneme, že $f \in \Theta(g)$, označujeme také jako $f(n) \in \Theta(g(n))$, právě tehdy, když platí $g \in O(f)$ a $f \in O(g)$ neboli

$$(\exists c_1 > 0)(\exists c_2 > 0)(\exists n_0 \geq 0)(\forall n \geq n_0) : 0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n).$$

Definice

Pro libovolné dvě funkce $f, g : \mathbb{N} \rightarrow \mathbb{N}$ řekneme, že $f \in o(g)$, označujeme také jako $f(n) \in o(g(n))$, právě tehdy, když platí

$$(\forall c > 0)(\exists n_0 > 0)(\forall n \geq n_0) : 0 \leq f(n) < c g(n).$$

Definice

Pro libovolné dvě funkce $f, g : \mathbb{N} \rightarrow \mathbb{N}$ řekneme, že $f \in \omega(g)$, označujeme také jako $f(n) \in \omega(g(n))$, právě tehdy, když platí

$$(\forall c > 0)(\exists n_0 > 0)(\forall n \geq n_0) : 0 \leq c g(n) < f(n).$$

Poznámka: Často se píše $f(n) = O(g(n))$ místo $f(n) \in O(g(n))$

Příklad: Při analýze složitosti vyhledávání čísla v posloupnosti jsme dostali $t_A(n) = 6n - 5$. Platí $t_A \in O(n)$. Takže můžeme říct, že složitost našeho algoritmu je v $O(n)$.

Příklad:

$$n \in O(n^2)$$

$$n^3 \in O(n^4)$$

$$1000n \in O(n)$$

$$0.00001n^2 - 10^{10}n \in \Theta(10^{10}n^2)$$

$$2^{\log_2 n} \in \Theta(n)$$

$$n^3 - n^2 \log^3 n + 1000n - 10^{100} \in \Theta(n^3)$$

$$n \in o(n^2)$$

$$n^3 + 1000n - 10^{100} \notin o(n^3)$$

$$n^2 \in \omega(n)$$

$$n^3 + n^2 \in \Omega(n^2)$$

$$n^3 + 2^n \in \Omega(n^2)$$

$$n! \in \Omega(2^n)$$

- Ne vždy je algoritmus s lepší asymptotickou složitostí výhodnější v praxi
- Můžeme si to ilustrovat na algoritmech pro třídění

| Algoritmus | Nejhorší | Průměrný |
|------------|---------------|---------------|
| Bubblesort | $O(n^2)$ | $O(n^2)$ |
| Heapsort | $O(n \log n)$ | $O(n \log n)$ |
| Quicksort | $O(n^2)$ | $O(n \log n)$ |

- Quicksort má horší asymptotickou složitost v nejhorším případě než heapsort, stejnou asymptotickou složitost v průměrném případě a přesto je v praxi nejrychlejší

- Ne vždy je vhodné uvažovat složitost jako funkci počtu znaků abecedy potřebných pro zápis vstupu
- Pro grafy často bývá složitost vyjádřena jako funkce dvou parametrů: m (počet hran) a n (počet vrcholů)
- Pro operace s posloupnostmi čísel (třídění, vyhledávání,...) bývá n počet čísel a ne cifer nebo bitů
- Pro rozhodování o prvočíselnosti nemůžeme brát $n = 1$ (vstupem je jedno číslo), ale uvažujeme počet bitů (popřípadě cifer)
- Při práci s maticemi uvažujeme většinou n jako počet řádků a m počet sloupců (popř. jen n je-li sloupců a řádku přibližně stejně) a ne přímo počet prvků matice

- O algoritmu s časovou složitostí $t_{\mathcal{A}}(n)$ řekneme, že je:
 - logaritmický**, pokud $t_{\mathcal{A}}(n) \in \Theta(\log n)$
 - lineární**, pokud $t_{\mathcal{A}}(n) \in \Theta(n)$
 - kvadratický**, pokud $t_{\mathcal{A}}(n) \in \Theta(n^2)$
 - kubický**, pokud $t_{\mathcal{A}}(n) \in \Theta(n^3)$
 - polynomiální**, pokud $t_{\mathcal{A}}(n) \in \Theta(n^k)$ pro nějaké $k \in \mathbb{N}$
 - exponenciální**, pokud $t_{\mathcal{A}}(n) \in \Theta(k^n)$ pro nějaké $k \in \mathbb{N}$
- Pro exponenciální složitost se používá také $2^{\Theta(n)}$, protože potom již nemusíme uvažovat různé základy mocniny.
- Pro logaritmickou složitost nehraje roli základ logaritmu, protože $\log_k n \in \Theta(\log_l n)$ pro všechna $k, l > 0$.

- Zatím jsme uvažovali, že provedení všech instrukcí trvá stejně dlouho
- V praxi mohou být některé instrukce časově náročnější
- Místo počtu instrukcí v nejhorším případě nás může zajímat počet provedení jedné konkrétní instrukce v nejhorším případě
- Pokud známe časy provedení jednotlivých instrukcí, můžeme si spočítat počty jejich provedení a čas běhu potom dostaneme jako $\sum_i n_i t_i$, kde n_i je počet provedení instrukce i a t_i je čas potřebný k vykonání této instrukce

Prostorová (paměťová) složitost algoritmů

- Zatím jsme se zajímali o čas, který potřebujeme k výpočtu
- Často bývá kritickou velikost paměti

Definice

Velikost paměti stroje RAM \mathcal{M} na vstupu x rozumíme počet buněk paměti, které stroj \mathcal{M} během svého výpočtu navštíví.

Definice

Prostorová složitost stroje RAM \mathcal{M} (v nejhorsím případě) je funkce $s_{\mathcal{M}} : \mathbb{N} \rightarrow \mathbb{N}$, kde $s_{\mathcal{M}}(n)$ udává maximální velikost paměti ze všech výpočtů nad vstupy x délky n .

Prostorová (paměťová) složitost algoritmů

- Pro konkrétní problém můžeme mít dva algoritmy takové, že jeden má menší prostorovou složitost a druhý zase časovou složitost
- Prostorová složitost je někdy kritičtější než časová. Když dojde paměť, ve výpočtu se nedá pokračovat
- Je-li časová složitost algoritmu v $O(f(n))$ je i prostorová v $O(f(n))$ (počet buněk navštívených RAMem nemůže být větší než počet kroků, protože v každém kroku navšíví maximálně jednu buňku)

Složitost problémů

- Zřejmě jsou některé problémy „složitější“ než jiné
- Jestliže pro nějaký problém známe algoritmus v $\Theta(n^3)$, který jej řeší, máme horní odhad pro složitost problému. Ale může existovat algoritmus v $\Theta(n^2)$, který daný problém také řeší.

Definice

Pro funkci $f : \mathbb{N} \rightarrow \mathbb{N}$ rozumíme **třídou časové složitosti** $\mathcal{T}(f)$ (nebo $\mathcal{T}(f(n))$) množinu těch problémů, které jsou řešeny RAMy s časovou složitostí v $O(f(n))$.

Definice

Pro funkci $f : \mathbb{N} \rightarrow \mathbb{N}$ rozumíme **třídou prostorové složitosti** $\mathcal{S}(f)$ (nebo $\mathcal{S}(f(n))$) množinu těch problémů, které jsou řešeny RAMy s prostorovou složitostí v $O(f(n))$.

- Pokud chceme ukázat, že problém je v $\mathcal{T}(f(n))$ stačí najít algoritmus se složitostí v $O(f(n))$ řešící daný problém.
- Pokud chceme ukázat, že problém není v $\mathcal{T}(f(n))$, musí ukázat, že neexistuje žádný algoritmus se složitostí v $O(f(n))$ řešící daný problém. To je obvykle dost obtížné.
- Třídy složitosti jsou závislé na zvoleném modelu (v našem případě stroj RAM). Pro jiný model bychom mohli dostat jiné třídy.

Definice

$$\text{PTIME} = \bigcup_{k=0}^{\infty} \mathcal{I}(n^k)$$

- Třída PTIME se ukazuje jako vhodná aproximace zvládnutelných problémů
- Není to ale přesné, algoritmus v $\Theta(n^{1000000})$ se moc za rychlý považovat nedá. Naopak i algoritmy s exponenciální složitostí mohou být použitelné, pokud nejsou určeny pro velké vstupy.
- Pokud je znám pro nějaký problém polynomiální algoritmus, často se najde i polynomiální algoritmus s nízkým stupněm polynomu (řekněme menším než 6)
- Třída PTIME není závislá na zvoleném (rozumném) modelu

Složitosti konkrétních problémů

Mějme dáno pole A []

- Nalezení konkrétního prvku v A lineárním prohledáváním je v $O(n)$.
- Časová složitost setřídění A je $O(n \log n)$.
- V setříděném poli A lze nalézt prvek binárním vyhledáváním v čase $O(\log n)$.

Vezměme aritmetiku dlouhých n -místných čísel.

- Sečíst dvě taková čísla lze v čase $O(n)$ a vynásobit v čase $O(n^2)$ běžnými “školními” postupy.
- Sofistikovaný Strassenův algoritmus vynásobí dvě n -místná čísla v čase $O(n \log n \log \log n)$.

Mějme daný graf G s n vrcholy a m hranami ($m = O(n^2)$).

- Komponenty souvislosti G najdeme v čase $O(n + m)$.
- Nejkratší cestu mezi dvěma vrcholy vypočteme v čase $O(n + m \log n)$.
- Minimální kostru nalezneme hladově v čase $O(n + m \log n)$.
- Rovinné nakreslení grafu lze nalézt (pokud existuje) v čase $O(n)$.