Zložitosť algoritmov

Bc. Katarína Olejková

KATEDRA INFORMATIKY

UNIVERZITA PALACKÉHO V OLOMOUCI

Zložitosť algoritmov

- Miera, podľa ktorej môžeme posudzovať efektivitu algoritmov, a to na základe:
  - Ako dlho trvá výpočet – **Časová zložitosť**
  - Koľko pamäte zaberie výpočet – **Pamäťová zložitosť**

Časová zložitosť

- Závisí od veľkosti vstupu
- Funkcia, ktorá veľkosti vstupu priradí trvanie výpočtu

- Veľkosť vstupu rozumieme, ako:
  - Hodnota čísla
  - Dĺžka textu
  - Počet prvkov v poli
  - Počet cifier
  - Počet uzlov v grafe
  - Atď..

- Trvanie výpočtu rozumieme, ako počet výpočetných krokov algoritmu (inštrukcií)
  - Trvanie výpočtu **nemeriame** skutočným časom (napr. počtom sekúnd) lebo ten je závislý od HW počítača

Časová zložitosť

- Príklad: budeme mať algoritmus A, ktorý na vstupe dostane ľubovoľné pole a výstupom bude vzostupne zoradené pole
  - Budeme mať vstupy algoritmu $I_1, I_2 .. I_m$ každý vstup má veľkosť n

    n = 5

    obmedzíme sa iba na tieto 3 vstupy (v skutočnosti ich bude oveľa viac pre takéto n)

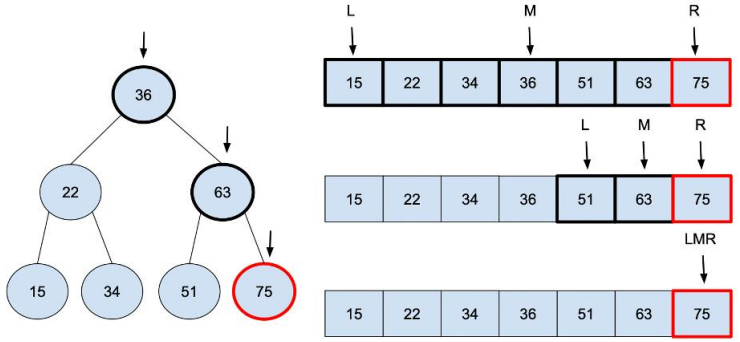
    I_1 = [1, 2, 3, 4, 5], I_2 = [1, 2, 4, 3, 5], I_3 = [5, 4, 2, 3, 1]

  - Povedzme, že výpočet bude trvať toľko, koľko prvkov vo vstupe je na nesprávnej pozícií

    $t_A(I_1)$ = 0, $t_A(I_2)$ = 2, $t_A(I_3)$ = 5

- Časová zložitosť v **najhoršom prípade**: max z množín $t_A(I_j)$ T(n) = 5
- Časová zložitosť v **priemernom prípade**: súčet množín $t_A(I_j)$ / m T(n) = 7 / 3

Časová zložitosť



- Často sa vyskytujúce časové zložitosti T(n):
  - Konštantná – vrátenie prvého prvku v poli
  - Logaritmická – binárne vyhľadávanie
  - Lineárna - nájdenie maxima v poli
  - Logaritmicko lineárna – merge-sort, quick-sort..
  - Kvadratická – dvojitý vnorený cyklus (insert-sort, select-sort)
  - Kubická – trojitý vnorený cyklus
  - Exponenciálna – rekurzívny výpočet Fibonacciho postupnosti
  - Faktoriál – výpočet permutácií poľa

Príklad 1 - Zapíšte algoritmus JeKladne(n) pomocou pseudokódu a určite jeho časovú zložitosť

- Problém: Zistite, či je dané celé číslo kladné
- Vstup: ľubovolné celé číslo **n**
- Výstup: “áno“/ “nie“

- Algoritmus:
  - Ak **n** > 0 zapíš na výstup “áno“
  - Inak zapíš na výstup “nie“

- Riešenie:

```
1. JeKladne(n)
2.     if n > 0
3.         print(“áno“)
4.     else
5.         print(“nie“)
```

Príklad 2 – Navrhnite algoritmus SudeNeboLiche(n), zapíšte ho pomocou pseudokódu a určite jeho časovú zložitosť

- Problém: Zistite, či je dané celé číslo sudé alebo liché
- Vstup: ľubovolné celé číslo **n**
- Výstup: “sudé“/“liché“

- Riešenie:

```
1. SudeNeboLiche(n)
2.     if n mod 2 = 0
3.         print(“sudé“)
4.     else
5.         print(“liché“)
```

Príklad 3 – Navrhnite algoritmus Signum(n), zapíšte ho pomocou pseudokódu a určite jeho časovú zložitosť

- Problém: Vypočítajte funkciu signum pre dané celé číslo
- Vstup: ľubovolné celé číslo **n**
- Výstup: 1/0/-1

Signum(n) = 1 pre n > 0
0 pre n = 0
-1 pre n < 0

- Riešenie:

```
1. Signum(n)
2.     if n > 0
3.         return 1
4.     else if n = 0
5.         return 0
6.     else
7.         return - 1
```

Príklad 4 – Navrhnite algoritmus Sucet(n), zapíšte ho pomocou pseudokódu a určite jeho časovú zložitosť

- Problém: Pre zadané celé číslo n vypočítajte súčet všetkých čísel od 1 do n
- Vstup: ľubovolné celé číslo **n**
- Výstup: súčet

- Riešenie:

```
1. Sucet(n)
2.     sum ← 0
3.     for i ← 1 to n
4.         sum ← sum + i
5.     return sum
```

Príklad 5 – Navrhnite algoritmus Nasobilka(n), zapíšte ho pomocou pseudokódu a určite jeho časovú zložitosť

- Problém: Pre zadané celé číslo n vypíšte násobilku od 1 * 1 až n * n
- Vstup: ľubovolné celé číslo **n**
- Výstup: výpis násobilky

Napr. pre n = 3 očakávaný výstup:
1 * 1 = 1
1 * 2 = 2
1 * 3 = 3
2 * 1 = 2
2 * 2 = 4
2 * 3 = 6
3 * 1 = 3
3 * 2 = 6
3 * 3 = 9

Príklad 6 – Navrhnite algoritmus SucetPole(A[0..n-1]), zapíšte ho pomocou pseudokódu a určite jeho časovú zložitosť

- Problém: Pre zadané pole A[0..n-1] vypočítajte súčet všetkých jeho prvkov
- Vstup: Pole A[0..n-1] (pole A s n prvkami)
- Výstup: súčet

Príklad 7 – Navrhnite algoritmus AritmetickyPrumer(A[0..n-1]), zapíšte ho pomocou pseudokódu a určite jeho časovú zložitosť

- Problém: Pre zadané pole A[0..n-1] vypočítajte aritmetický priemer jeho prvkov
- Vstup: Pole A[0..n-1] (pole A s n prvkami)
- Výstup: priemer