Algoritmus, algoritmická složitost

Algoritmus, algoritmická složitost, rekurze, náhodnost

Joseph Soukup – P4

Algoritmus

- Je schématický postup pro řešení určitého druhu problémů, který je prováděn pomocí konečného množství přesně definovaných kroků.
- Ne pouze v programování (například i recepty)
- **Vlastnosti**:
    - **Konečnost** – algoritmus má konečné množství kroků
    - **Určitost** – všechny kroky jsou přesně definovány
    - **Korektnost** – algoritmus skončí pro libovolná (korektní data správným výsledkem v konečném množství kroků)
    - **Obecnost** – algoritmus řeší všechny úlohy daného typu

Prostorová složitost

- Pro konkrétní problém můžeme mít dva algoritmy takové, že jeden má menší prostorovou složitost a druhý zase časovou složitost
- Je-li časová složitost algoritmu O(f(n)) je i prostorová O(fn(n)) (počet použitých paměťových buněk nemůžeme být větší než počet provedených operací, protože v každém kroku se použije jen nějaký omezený počet buněk (omezený nějakou konstantou nezávislou na velikosti vstupu))
- Prostorová složitost může být mnohdy o dost menší než časová složitost - například paměťová složitost algoritmu „Insertion-sort“ jen O(n), zatímco časová O(n**2)

Algoritmická složitost

- Algoritmická složitost se zabývá tím, jestli je algoritmus schopen skončit v „rozumném čase“
- Rozumným časem se rozumí čas takový, který umožňuje výsledek smysluplně využít
- Pomocí algoritmické složitosti se určuje, kde je vhodné, jaký algoritmus použít s porovnání s jinými
- **Porovnání podle:**
    - Doby výpočtu podle daného algoritmu potřebná pro zpracování daného objemu dat (časová složitost)
    - Velikost paměti využívané při výpočtu (paměťová složitost)
- Pro složitost se používá symbol O()
- Příklad výpočetní náročnosti, kdy jedna operace trvá jednu nanosekundu:
    -

Asymptotická složitost	Velikost vstupních dat							
	10	20	50	100	1 000	1 000 000	1 000 000 000	10^{20}
$\log \log n$	2 ns	3 ns	3 ns	3 ns	4 ns	5 ns	5 ns	7 ns
$\log n$	4 ns	5 ns	6 ns	7 ns	10 ns	20 ns	30 ns	67 ns
$\sqrt{n}$	4 ns	8 ns	8 ns	10 ns	32 ns	1 µs	32 µs	10 s
n	10 ns	20 ns	50 ns	100 ns	1 µs	1 ms	1 s	3 171 let
$n \log n$	34 ns	87 ns	283 ns	665 ns	10 µs	20 ms	30 s	210 675 let
n^2	100 ns	400 ns	3 µs	10 µs	1 ms	16 min 40 s	32 let	
n^3	1 µs	8 µs	125 µs	1 ms	1 s	32 let		
n^4	10 µs	160 µs	6 ms	100 ms	16 min 40 s	31 688 088 let		
2^n	1 µs	1 ms	13 dní	40×10^{12} let				
3^n	59 µs	4 s	22 760 000 let					
$n!$	4 ms	77 let						
n^n	10 s	$3{,}32 \times 10^9$ let						
2^{2^n}	$5{,}7 \times 10^{291}$ let							

- **Výpočet:**

- Koukneme se na každý řádek, kolikrát se provede, než se problém vyřeší
- Příklad:

```
int MAX = 0, i = 0;
while (i <= n-1) {
    if (A[i] > MAX)
        MAX = A[i];
    i = i + 1;
}
return MAX
```

  - Složitost:
    - 1. řádek se provede 1 (inicializace proměnných)
    - 2. řádek se provede n-krát vždy musíme projít celé pole, prvek po prvku, proto si nemůžeme dovolit se na nějaký prvek nepodívat
    - 3. řádek se provede n-krát prvek se vždy porovnává s MAX
    - 4. řádek se provede minimálně 0krát, maximálně n-krát -> provádí se jen pokud je MAX menší než prvek
    - 5. řádek se provede n-krát index se vždy zvětší
    - 6.řádek se provede 1krát – konec bloku
    - 7. řádek se provádí 1krát – na konci funkce se vrátí jednou hodnotu MAX
    - Nyní vypočítáme nejlepší případ a nejhorší případ.
    - **Nejlepší:**
      - 1+n+n+0+n+1+1 = 3n + 3 = 0(n)
    - **Nejhorší:**
      - 1+n+n+n+n+1+1 = 4n + 3 = 0(n)

Dělení algoritmů

- Iterativní algoritmus -> opakování určité své části (bloku)
- Rekurzivní algoritmus -> opakuje kód prostřednictvím volání sama sebe (lze převést do iterativní podoby)
- Převod řeší automatický kompilátor nebo virtuální stroj daného programovacího jazyka
- Výhoda rekurzivních algoritmů:
  - Snadná čitelnost
  - Kompaktní zápis
- Nevýhoda:
  - Spotřrba dodatečných systémových prostředků pro udržení jednotlivých rekurzivních volání
- **Iterativní –** bubble sort, insertion sort
- **Rekurzivní –** merge sort, quicksort

Deterministické a nedeterministické algoritmy

- Deterministický -> v každém kroku jedna možnost
- Nedeterministický -> více
- Př. Deterministický a nedeterministický automat

Sériové, paralelní a distribuované algoritmy

- Sériový algoritmus vykonává všechny kroky v sérii (jeden po druhém)
- Paralelní algoritmus tyto kroky vykonává zároveň (ve více vláken)
- Distribuovaný algoritmus kroky vykonává zároveň na více strojích

Rekurze

- Stav, kdy je určitý objekt v nějakém smyslu součástí sebe samotného
- Určitá procedura nebo funkce znovu volána dříve, než je dokončeno její předchozí volání
- Při optimalizaci programu se většinou snažíme rekurzi vyhnout
- Volání může probíhat přímo nebo nepřímo:
    - **Přímá rekurze:**
        - Nastává, když podprogram volá přímo sám sebe
    - **Nepřímá rekurze:**
        - Je situace kdy vzájemné volání podprogramů vytvoří „kruh“ (např. ve funkci A se volá funkce B a ve funkci B se volá opět funkce A.)
- Podprogram může být volán jednou nebo vícekrát:
    - **Lineární rekurze:**
        - Volá sama sebe pouze jednou. Vytváří se takto lineární struktura postupně volaných podprogramů
    - **Stromová rekurze:**
        - Podprogram se v rámci jednoho vykonaní svého úkolu vyvolá vícekrát.
- Může mít velkou spotřebu paměti
- Nejčastěji se používá např. ke kontrole, zda výstupní parametry odpovídají stanoveným podmínkám

Náhodnost

- Náhodné algoritmy se snaží nalézt řešení problému náhodným rozhodováním o svém postupu
- Při vytváření musíme dokončit všechny směry, kterými se může program vydat
- Program se rozhoduje několika způsoby:
    - V každém uzlu rozhodne postup náhodně
    - Na začátku vybere jeden z deterministických algoritmů
- Generování pseudonáhodných čísel – Kongruentní generátor – zdánlivě náhodná čísla
- Pravý generátor náhodných čísel – využívá náhodnost získanou z fyzikálních jevů