Algoritmy a datové struktury

V počítačové vědě platí základní rovnice: **Program = Algoritmy + Datové struktury**. Algoritmus definuje postup řešení problému, zatímco datová struktura určuje, jakým způsobem budou data uložena v paměti pro co nejefektivnější přístup.

1. Datové struktury: Organizace dat

Volba správné struktury může zrychlit program tisícinásobně. Rozdělujeme je na základní (lineární) a pokročilé (nelineární).

Lineární struktury

* **Pole (Array):** Kolekce prvků stejného typu uložených v souvislém bloku paměti. Přístup k prvku je bleskový, ale změna velikosti pole je náročná. * **Spojový seznam (Linked List):** Prvky jsou rozesety v paměti a každý prvek ukazuje na ten následující. Snadno se do něj vkládá, ale hledání v něm je pomalé. * **Zásobník (Stack):** Princip **LIFO** (Last In, First Out). Jako stoh talířů – poslední položený berete jako první. * **Fronta (Queue):** Princip **FIFO** (First In, First Out). Jako fronta v obchodě – kdo dřív přijde, je dřív obsloužen.

Nelineární struktury

* **Strom (Tree):** Hierarchická struktura (např. souborový systém). Nejdůležitější je **Binární vyhledávací strom**, který umožňuje velmi rychlé hledání. * **Graf (Graph):** Množina uzlů propojených hranami. Používá se pro modelování sociálních sítí nebo map (navigace). * **Hešovací tabulka (Hash Table):** Umožňuje vyhledat hodnotu pomocí klíče v téměř konstantním čase.

[Image of Binary Search Tree and Graph data structure diagrams]

—

2. Algoritmy: Postupy řešení

Algoritmus je konečná posloupnost přesně definovaných instrukcí pro vyřešení daného úkolu.

Klíčové vlastnosti algoritmu:

- **Konečnost:** Musí vždy skončit po určitém počtu kroků.
- **Determinovanost:** Pro stejné vstupy musí vždy vydat stejné výstupy.
- **Obecnost:** Neměl by řešit jen jeden konkrétní případ, ale celou třídu úloh.

Typické příklady algoritmů:

- **Třídění (Sorting):** Seřazení dat (např. QuickSort, MergeSort).
- **Vyhledávání (Searching):** Např. Binární vyhledávání v seřazeném poli.
- **Grafové algoritmy:** Hledání nejkratší cesty (Dijkstrův algoritmus).

—

3. Asymptotická složitost (Velké O)

Abychom mohli porovnat, který algoritmus je lepší, používáme zápis **Big O notation**. Ten neříká, kolik sekund program poběží, ale jak poroste náročnost se zvyšujícím se množstvím dat (n).

Složitost	Název	Příklad
$O(1)$	Konstantní	Přístup k prvku v poli pomocí indexu.
$O(\log n)$	Logaritmická	Binární vyhledávání (při každém kroku půlíme zbytek dat).
$O(n)$	Lineární	Prohledání neseřazeného seznamu (musíme projít vše).
$O(n \log n)$	Linearitmetická	Efektivní třídicí algoritmy (MergeSort).
$O(n^2)$	Kvadratická	Neefektivní třídění (Bubble Sort - dva cykly v sobě).

—

4. Strategie návrhu algoritmů

- **Rozděl a panuj (Divide and Conquer):** Rozdělení problému na menší podproblémy (např. MergeSort).
- **Dynamické programování:** Rozkládání na podproblémy, jejichž výsledky si pamatujeme, abychom je nepočítali znovu (např. Fibonacciho posloupnost).
- **Hladové algoritmy (Greedy):** V každém kroku se vybere lokálně nejlepší řešení v naději, že povede k globálnímu optimu.

Související články:

- Programovací jazyky (Java, Python, C++)
- Objektově orientované programování
- Fyzikální limity výpočtů

Tagy: dev algorithms data_structures complexity sorting binary_tree