C2142 Návrh algoritmů pro přírodovědce

10. Přístupy k řešení problémů I.

Tomáš Raček

Hrubá síla

Hrubá síla (brute force) představuje přímočarý přístup založený na definici problému.

Vlastnosti

- obecně výpočetně nejnáročnější postup → použitelné pouze pro velmi malé instance problémů
- v případě optimalizačních úloh se jedná o zkoušení všech potenciálních řešení
- pro některé typu problémů jediná možnost

Příklady

- Selection sort
- naivní násobení matic
- jednoduché hledání podřetězce v textu

Rekurzivní algoritmy

Myšlenka rekurzivních algoritmů spočívá v znovupoužití algoritmu na problém menší velikosti. Speciální (triviální) případy řešíme přímo.

```
def fact_recursive(n):
    return n * fact_recursive(n - 1) if n > 0 else 1


def fact_iterative(n):
    result = 1
    for i in range(1, n + 1):
        result *= i

    return result
```

Poznámka. Volání funkce je ve srovnání s iterací cyklu drahá operace. Vysoká úroveň rekurzivního zanoření navíc může narazit na limity velikosti zásobníku.

Hanoiské věže

Hanoiské věže jsou hlavolam, v rámci kterého je potřeba přemístit všechny disky z jednoho kolíku na jiný za dodržení dalších pravidel.

Poznámka. Existuje velmi jednoduché rekurzivní řešení.

Složitost řešení problému Hanoiských věží je exponenciální vůči počtu disků (přesně $2^n - 1$ operací).

Rozděl a panuj

Rozděl a panuj (divide and conquer, D & C) je přístup vycházející z rekurzivních algoritmů založený na dělení problému na menší, snadněji zvládnutelné, podproblémy.

Princip

1. Rozděl problém na menší podproblémy
   - stejného typu
   - nepřekrývající se
2. Rekurzivně vyřeš každý podproblém
3. Zkombinuj řešení podproblémů v řešení původního problému

Příklady

- Mergesort
- Quicksort

Master theorem

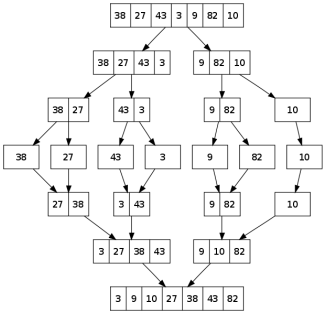
Master theorem je nástroj pro určování složitosti algoritmů založených na přístupu rozděl a panuj. Pokud je velikost podproblémů shodná, lze využít následujícího vztahu:

$$T(n) = aT(n/b) + O(n^d)$$

- $T(n)$ – počet kroků nutných pro vyřešení problému o velikosti n
- a – počet podproblémů
- b – faktor určující velikost podproblémů
- n^d – rozdělení na podproblémy | sloučení řešení jednotlivých podproblémů

$$T(n) = \begin{cases} O(n^d) & \text{pro } d > \log_b a \\ O(n^d \log n) & \text{pro } d = \log_b a \\ O(n^{\log_b a}) & \text{pro } d < \log_b a \end{cases}$$

Master theorem – Mergesort



Složitost Mergesortu. Každé pole je rozděleno na dvě části o poloviční velikosti ($a = b = 2$), slévání seřazených podposloupností je lineární operace ($d = 1$). Platí $d = \log_b a \rightarrow T(n) = O(n^d \log n)$.

$$T(n) = 2T(n/2) + O(n) \rightarrow T(n) = O(n \log n)$$

Násobení matic

Úkol. Popište algoritmus pro násobení dvou matic ($Z = XY$) o rozměrech $n \times n$.

Naivní násobení matic představuje učebnicový způsob řešení se složitostí $O(n^3)$.

$$\forall (i, j) \in \{1..n\} \times \{1..n\} : Z_{i,j} = \sum_{k=1}^{n} X_{i,k} Y_{k,j}$$

Blokové násobení matic je přístup, kdy každou z matic rozdělíme na menší a násobíme dle následujícího schématu:

$$X = \begin{pmatrix} A & B \\ C & D \end{pmatrix} \qquad Y = \begin{pmatrix} E & F \\ G & H \end{pmatrix}$$

$$Z = XY = \begin{pmatrix} A & B \\ C & D \end{pmatrix} \cdot \begin{pmatrix} E & F \\ G & H \end{pmatrix} = \begin{pmatrix} AE + BG & AF + BH \\ CE + DG & CF + DH \end{pmatrix}$$

Násobení matic

Složitost blokového násobení matic je však zřejmě stále shodná s naivním přístupem, tedy $O(n^3)$.

Poznámka. V reálném běhu je dekompozice na bloky výrazně rychlejší řešení díky lepšímu využití vyrovnávací paměti.

Rekurzivní algoritmus násobení matic

$$Z = XY = \begin{pmatrix} A & B \\ C & D \end{pmatrix} \cdot \begin{pmatrix} E & F \\ G & H \end{pmatrix} = \begin{pmatrix} AE + BG & AF + BH \\ CE + DG & CF + DH \end{pmatrix}$$

- aplikujme stejný (D & C) přístup pro výpočet součinů $AE, BG, \ldots$
- aplikací MT získáváme

$$T(n) = 8T(n/2) + O(n^2) \rightarrow T(n) = O(n^3)$$

Strassenův algoritmus

Myšlenka. Využijeme rekurzivní algoritmus pro násobení matic, avšak pomocí algebraických transformací snížíme počet nutných násobení. Počet sčítání není příliš významný.

$$\begin{aligned} P_1 &= A(F-H) & P_5 &= (A+D)(E+H) \\ P_2 &= (A+B)H & P_6 &= (B-D)(G+H) \\ P_3 &= (C+D)E & P_7 &= (A-C)(E+F) \\ P_4 &= D(G-E) \end{aligned}$$

$$Z = XY = \begin{pmatrix} P_5 + P_4 - P_2 + P_6 & P_1 + P_2 \\ P_3 + P_4 & P_1 + P_5 - P_3 - P_7 \end{pmatrix}$$

Složitost Strassenova algoritmu

$$T(n) = 7T(n/2) + O(n^2) \rightarrow T(n) = O(n^{\log_2 7}) \approx O(n^{2,81})$$

Hladové algoritmy

Hladový algoritmus (greedy algorithm) v každém kroce výpočtu výbírá aktuálně nejlepší možnost, přičemž spoléhá, že tato posloupnost voleb vede ke globálně nejlepšímu řešení.

Vlastnosti

- ne vždy lze s úspěchem použít – pouze některé problémy mají tuto strukturu
- časově efektivní

Příklady

- Kruskalův a Primův algoritmus
- Dijkstrův algoritmus

Minimální počet mincí

Problém. Vyplaťte zadanou částku pomocí minimálního počtu mincí.

Příklad. Částka 79, hodnoty mincí 1, 2, 5, 10,...

- hladový přístup – volím vždy minci nejvyšší hodnoty menší než zbývající částka
- 79 = 50 + 20 + 5 + 2 + 2

Příklad. Částka 6, hodnoty mincí 1, 3 a 4.

- hladový přístup – 6 = 4 + 1 + 1
- optimální řešení – 6 = 3 + 3

Příklad. Částka 14, hodnoty mincí 3, 7 a 10.

- optimální řešení – 14 = 7 + 7
- hladový přístup – 14 = 10 + ?