Budowa algorytmów - dziel i zwyciężaj

Wstęp do Informatyki i Programowania

Maciek Gębala

12 grudnia 2024

Notatki

Zasada dziel i zwyciężaj

Zasada dziel i zwyciężaj (ang. divide and conquer) jest bardzo efektywną metodą projektowania algorytmów.

Dzieli ona rozwiązywany problem na pewną liczbę prostszych podproblemów (najczęściej z rozłącznymi danymi) a następnie rozwiązuje, zgodnie z tą zasadą, każdy z podproblemów i na koniec przeprowadzana jest synteza rozwiązania problemu z rozwiązań podproblemów.

Bardzo często wykorzystuje się w tych algorytmach rekurencję.

Notatki

Zasada dziel i zwyciężaj

Pseudokod zasady dziel i zwyciężaj

```
function RozwiążProblem(P)
    if P jest prostym problemem then
        return RozwiązanieProblemu(P)
    else
        podziel P na prostsze podproblemy P_1, P_2, ..., P_k   ▷ analiza
        for i ← 1 ... k do
            R_i ← RozwiążProblem(P_i)                          ▷ rekurencja
        end for
        return Synteza(R_1, R_2, ..., R_k)                     ▷ synteza
    end if
end function
```


Notatki

Złożoność obliczeniowa

Niech n będzie rozmiarem problemu P i załóżmy, że problem P jest dzielony na k podproblemów m razy mniejszego rozmiaru.

Niech funkcja $T(n)$ wyraża liczbę operacji jakie wykonywane są podczas rozwiązywania problemu P zgodnie z zasadą dziel i zwyciężaj. Wówczas:

$$T(n) = \begin{cases} O(1) & \text{dla } n \leqslant c \\ k \cdot T(\frac{n}{m}) + f(n) & \text{dla } n > c \end{cases}$$

tzn. rozwiązanie prostego problemu wykonuje się w czasie stałym dla danych rozmiaru c, podział i synteza odbywa się w czasie $f(n)$.

Wartość funkcji $T(n)$ zależy od relacji między k i m oraz funkcji f.

Jak rozwiązywać takie równania będzie na przyszłych semestrach. Teraz możemy użyć WolframAlpha.

Notatki

Przykłady: binary search w uporządkowanej tablicy

Algorytm omówiony na ćwiczeniach.

Złożoność w liczbie porównań

$$T(n) = \begin{cases} 1 & \text{dla } n = 1 \\ T(\frac{n}{2}) + 1 & \text{dla } n > 1 \end{cases}$$

Co po rozwiązaniu da nam $T(n) \approx \log_2 n$.

Notatki

Przykłady: sortowanie przez scalanie (MergeSort)

Algorytm omówiony na ćwiczeniach.

Złożoność w liczbie porównań

$$T(n) = \begin{cases} 0 & \text{dla } n \leqslant 1 \\ 2 \cdot T(\frac{n}{2}) + O(n) & \text{dla } n > 1 \end{cases}$$

Co po rozwiązaniu da nam $T(n) \approx n \log_2 n$.

Notatki

Przykład: szybkie mnożenie liczb (algorytm Karacuby)

Dodawanie do siebie dwóch liczb n-bitowych wymaga $O(n)$ operacji bitowych. Analogicznie odejmowanie.

Proste mnożenie przez siebie dwóch liczb n-bitowych wymaga n operacji przesunięć bitowych i n operacji dodawania, czyli $O(n^2)$ operacji bitowych.

Załóżmy, że $n = 2m$. Wtedy liczbę n bitową $(a_{n-1} \dots a_0)_2$ można przedstawić jako sumę $(a_{2m-1} \dots a_m)_2 \cdot 2^m + (a_{m-1} \dots a_0)_2$, gdzie przemnożenie przez 2^m jest możliwe w czasie $O(m)$ przez prostą operację przesunięcia bitów.

Notatki

Przykład: szybkie mnożenie liczb (algorytm Karacuby)

Załóżmy teraz, że a jest liczbą $2m$ bitową przedstawioną jako dwie liczby m bitowe a_1 i a_0 (tzn. $a = a_1 \cdot 2^m + a_0$. Analogicznie, definiujemy b. Wtedy łatwo zauważyć, że

$$a \cdot b = a_1 \cdot b_1 \cdot 2^{2m} + (a_1 \cdot b_0 + a_0 \cdot b_1) \cdot 2^m + a_0 \cdot b_0$$

Czyli mamy algorytm typu dziel i zwyciężaj.

Złożoność bitową tego algorytmu można wyrazić wzorem

$$T(n) = \begin{cases} O(1) & \text{dla } n \leqslant 1 \\ 4 \cdot T(\frac{n}{2}) + O(n) & \text{dla } n > 1 \end{cases}$$

Co po rozwiązaniu daje $T(n) \approx O(n^2)$, czyli tyle samo co algorytm tradycyjny.

Notatki

Przykład: szybkie mnożenie liczb (algorytm Karacuby)

Karacuba zauważył, że liczby $a_1 \cdot b_1$, $a_1 \cdot b_0 + a_0 \cdot b_1$ i $a_0 \cdot b_0$ można wyliczyć wykonując trzy zamiast cztery mnożenia:

$$\begin{aligned} X &= a_0 \cdot b_0 \\ Y &= a_1 \cdot b_1 \\ Z &= (a_1 + a_0) \cdot (b_1 + b_0) \end{aligned}$$

Wtedy

$$a_1 \cdot b_0 + a_0 \cdot b_1 = Z - (X + Y)$$

Teraz łatwo zbudować algorytm metodą dziel i zwyciężaj, który będzie miał złożoność bitową określoną wzorem:

$$T(n) = \begin{cases} O(1) & \text{dla } n \leqslant 1 \\ 3 \cdot T(\frac{n}{2}) + O(n) & \text{dla } n > 1 \end{cases}$$

Co po rozwiązaniu daje $T(n) \approx O(n^{\log_2 3}) \approx O(n^{1.585})$, czyli dużo lepiej niż $O(n^2)$.

Notatki

Przykład: szybkie sortowanie (QuickSort)

Pseudokod

```
procedure QuickSort(A[1 : n], l, p)
    if l < p then                          ▷ Mamy co najmniej 2 elementy
        pivot ← A[p]                       ▷ Element dzielący
        i ← l
        for j ← l ... p − 1 do             ▷ Porządkowanie tablicy
            if A[j] ⩽ pivot then
                Swap(A[i], A[j])
                i ← i + 1
            end if
        end for
        Swap(A[i], A[p])
        QuickSort(A[1 : n], l, i − 1)
        QuickSort(A[1 : n], i + 1, p)
    end if
end procedure
```


Notatki

Przykłady: Szybkie sortowanie (QuickSort)

Złożoność w liczbie porównań

$$T(n) = \begin{cases} 0 & \text{dla } n \leqslant 1 \\ T(m) + T(n - m - 1) + O(n) & \text{dla } n > 1 \end{cases}$$

dla pewnego m takiego, że $0 \leqslant m < n$.

Nie wiemy jakie będzie te m, więc nie potrafimy ogólnie oszacować tego czasu.

Przypadek optymistyczny

Jeśli zawsze $m \approx \frac{n}{2}$, to $T(n) \approx O(n \log_2 n)$.

Przypadek pesymistyczny

Jeśli zawsze $m = 0$, to $T(n) \approx O(n^2)$.

Notatki

Notatki