NOTATKI I UZUPEŁNIENIA DO WYKŁADU 3

20 listopada 2016

Konstrukcja algorytmu: Zadanie rozmiaru n zostaje sprowadzone do zadania rozmiaru $\lfloor n/2 \rfloor$ + pewna stała liczba działań

Złożoność: $T(n)$=?

Zależność rekurencyjna dla funkcji $T(n)$

$$T(n) = T(n/2) + c$$

Na mocy twierdzenia

$$T(n) = \Theta(\lg n)$$

Każdy algorytm mający taką konstrukcję ma złożoność logarytmiczną

Konstrukcja algorytmu: Zadanie rozmiaru n zostaje sprowadzone do dwóch zadań rozmiaru $\lfloor n/2 \rfloor$ + pewna stała liczba działań

Złożoność: $T(n)$=?

Zależność rekurencyjna dla funkcji $T(n)$

$$T(n) = 2T(n/2) + c$$

Na mocy twierdzenia

$$T(n) = \Theta(n)$$

Każdy algorytm mający taką konstrukcję ma złożoność liniową

Również, gdy mamy stałą liczbę działań dla każdego z n elementów

Typeset by $\mathcal{A}\mathcal{M}\mathcal{S}$-TEX

Przykład 27. Dany jest wektor $A[1..n]$, taki że

$$A[1] \leq A[2] \leq \ldots \leq A[n]$$

Wyznaczyć indeks i składowej o zadanej wartości x (*Wyszukiwanie binarne*)

```
ELEMENT-4 (A, n, x)
   i = 1
   j = n
   repeat k = (i + j) div 2
           if x > A[k]
             then i = k + 1
             else  j = k − 1
        until (A[k] == x) or (i > j)
   if A[k] == x
     then return k
     else  pisz komunikat
```

$$W(n) = O(\lg n)$$

Przykład 28. *Potęga binarna*

```
POTĘGA-BINARNA (a, n)
   b = 1
   k = n
   c = a
   while k ≠ 0
         do if (k mod 2)==0
               then c = c ∗ c
                    k = k div 2
               else  b = b ∗ c
                    k = k − 1
   return b
```

$$T(n) = \Theta(\lg n)$$

Przykład 21 cd.

$$a_n = a_{n-1} + a_{n-2}, \qquad n \geq 2, \quad a_0 = 0, \quad a_1 = 1$$

Technika dziel i zwyciężaj

FIB (n)
   if $n == 0$
    then return 0
    else if $n == 1$
          then return 1
           else return FIB$(n-1)$+FIB$(n-2)$

$T(n)$ – liczba wywołań procedury FIB

$$T(n) = T(n-1) + T(n-2) + 1$$

Własność. Dla $n \geq 2$

$$T(n) > 2^{n/2}$$

Stosując notację asymptotyczną

$$T(n) = \Omega(2^{n/2})$$

Technika programowania dynamicznego

FIB-1 (n)
   $F[0] = 0$
   $F[1] = 1$
   for $k = 2$ **to** n
      do $F[k] = F[k-1] + F[k-2]$
   return $F[n]$

$$T(n) = \Theta(n)$$

Czy można jeszcze szybciej ?

$$T(n) = \Theta(\lg n)?$$

Przykład 25 cd. Wieże Hanoi

HANOI (n, X, Y, Z)
  if $n == 1$
    then $X \to Y$
    else HANOI $(n-1, X, Z, Y)$
      $X \to Y$
      HANOI $(n-1, Z, Y, X)$

Złożoność: $T(n)$ – liczba ruchów potrzebna do przeniesienia n krążków

Zależność rekurencyjna:

$$T(n) = 2T(n-1) + 1 \quad \text{dla } n \geq 2$$

Warunek początkowy $T(1) = 1$

Własność. Dla $n \geq 2$

$$T(n) = 2^n - 1$$

Złożoność wykładnicza

$$T(n) = \Theta(2^n)$$

Algorytmy sortowania

Dane: Tablica $A[1..n]$ zawierająca liczby

sortowanie przez wstawianie
sortowanie bąbelkowe
sortowanie przez scalanie
sortowanie szybkie
sortowanie przez kopcowanie
sortowanie przez zliczanie

Sortowanie przez wstawianie

Mając posortowaną tablicę elementów

$$A[1..j-1]$$

wstawiamy pojedynczy element $A[j]$ we właściwe miejsce tablicy, otrzymując większą posortowaną tablicę elementów $A[1..j]$

– elementy tablicy są sortowane **w miejscu**

– przykład **metody przyrostowej**

```
SORT-W (A, n)
    for j = 2 to n
        do r = A[j]
           i = j - 1
           while i > 0 and A[i] > r
                 do A[i + 1] = A[i]
                    i = i - 1
           A[i + 1] = r
```

Złożoność

t_j – liczba sprawdzeń warunku wejścia do pętli
while dla $j = 2, 3, \ldots, n$

$$T(n) = \sum_{j=2}^{n} t_j$$

Złożoność optymistyczna

Dane wejściowe są uporządkowane

$$t_j = 1 \quad \text{dla} \;\; j = 2, 3, \ldots, n$$

$$B(n) = \Theta(n)$$

Złożoność pesymistyczna

Dane wejściowe są w odwrotnym porządku

Należy porównać każdy element $A[j]$ z każdym elementem posortowanej już tablicy $A[1..j-1]$

$$t_j = j \quad \text{dla} \;\; j = 2, 3, \ldots, n$$

$$W(n) = \Theta(n^2)$$

Sortowanie bąbelkowe

Jeżeli przeglądamy tablicę liczb po kolei
i **zamieniamy miejscami** dwie sąsiednie liczby, gdy są w odwrotnym uporządkowaniu (tj. pierwsza jest większa od drugiej), to po zakończeniu przebiegu największa liczba znajdzie się na końcu tablicy

Instrukcja zamiany

$$a \leftrightarrow b$$

SORT-B (A, n)

```
for i = n downto 2
    do for j = 1 to i − 1
            do if A[j] > A[j + 1]
                  then A[j] ↔ A[j + 1]
```

$$T(n) = \Theta(n^2)$$

Wersja z wartownikiem

SORT-BW (A, n)

```
i = n
while i ≠ 0
      do k = 0
         for j = 1 to i − 1
             do if A[j] > A[j + 1]
                   then A[j] ↔ A[j + 1]
                        k = j
         i = k
```

$$B(n) = \Theta(n) \qquad W(n) = \Theta(n^2)$$

Sortowanie przez scalanie

– **dzielimy** tablicę na dwie podtablice

– **zwyciężamy** sortując każdą podtablicę, używając rekurencyjnie sortowania przez scalanie

– **łączymy** posortowane podtablice poprzez scalanie ich w jedną posortowaną tablicę

SORT-SCAL (A, p, r)

SORT-SCAL (A, p, r)
  if $p < r$
    then $q = (p + r)$ div 2
      SORT-SCAL (A, p, q)
      SORT-SCAL $(A, q + 1, r)$
      SCAL (A, p, q, r)

Mechanizm rekursji nie uruchamia się, gdy pozostaje do posortowania jeden element

Aby posortować całą tablicę $A[1..n]$, wywołujemy SORT-SCAL $(A, 1, n)$

Złożoność

Dziel: Znajdź środek przedziału – $\Theta(1)$

Zwyciężaj: Rozwiąż rekurencyjnie 2 podproblemy, każdy rozmiaru $n/2$ – daje w sumie ,,czas" $2T(n/2)$

Połącz: Procedura scalania dla n elementów działa w czasie $\Theta(n)$

$$T(n) = 2T(n/2) + \Theta(n)$$

$$T(n) = 2T(n/2) + \Theta(n)$$

Stosujemy twierdzenie o rekurencji uniwersalnej
$a = 2$, $b = 2$, $f(n) = \Theta(n)$

$$n^{\log_b a} = n$$

$$T(n) = \Theta(n \lg n)$$

tj. **złożoność logarytmiczno-liniowa**

Procedura scalania

$$\text{SCAL}\,(A, p, q, r)$$

Uporządkowane podciągi, które mają być scalone w jeden uporządkowany ciąg umieszczone są w tablicy A na miejscach, odpowiednio, od indeksu p do indeksu q i od indeksu $q+1$ do indeksu r, gdzie $p \leq q < r$. Scalony ciąg zostaje umieszczony w tablicy A na miejscach od $A[p]$ do $A[r]$

SCAL (A, p, q, r)
    $n = q - p + 1$
    $m = r - q$
    for $i = 1$ **to** n
        do $B[i] = A[p + i - 1]$
    for $j = 1$ **to** m
        do $C[j] = A[q + j]$
    $B[n + 1] = \infty$
    $C[m + 1] = \infty$
    $i = 1$
    $j = 1$
    for $k = p$ **to** r
        do if $B[i] \leq C[j]$
            then $A[k] = B[i]$
                $i = i + 1$
            else $A[k] = C[j]$
                $j = j + 1$