Notatki do wykładu 5

8 listopada 2018

Przykład 31. *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 23 cd.

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

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

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

Dowód (indukcja względem n). Dla $n = 2$ oczywiste, bo $T(2) = 3$. Załóżmy, że

$$T(m) > 2^{m/2} \quad \text{dla} \quad 2 \leq m < n$$

Z zależności rekurencyjnej

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

na mocy założenia indukcyjnego

$$\begin{aligned} T(n) &> 2^{(n-1)/2} + 2^{(n-2)/2} + 1 \\ &> 2^{(n-2)/2} + 2^{(n-2)/2} \\ &= 2^{n/2} \end{aligned}$$

Technika programowania dynamicznego

FIB-2 (n)
  $F0 = 0$
  $F1 = 1$
  for $k = 2$ **to** n
    do $Fk = F0 + F1$
      $F0 = F1$
      $F1 = Fk$
  return Fk

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

Czy można jeszcze szybciej ?

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

Przykład 29 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 $n \geq 2 \quad T(n) = 2^n - 1$

Złożoność wykładnicza

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

Dowód (indukcja względem n).
Dla $n = 1$ - prawda. Założenie indukcyjne

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

Na mocy rekurencji

$$\begin{aligned} T(n) &= 2T(n-1) + 1 \\ &= 2(2^{n-1} - 1) + 1 \\ &= 2^n - 1 \end{aligned}$$

Przykład 25 cd. NWD

$$NWD(a,b) = NWD(b, a \bmod b)$$

NWD (a,b)
  if $b == 0$
    then return a
    else return NWD$(b, a \bmod b)$

Własność. Jeżeli $a > b \geq 0$ oraz wywołanie NWD(a,b) prowadzi do $k \geq 1$ wywołań rekurencyjnych, to

$$a \geq F_{k+2} \quad \text{i} \quad b \geq F_{k+1}$$

Twierdzenie Lamego. Dla dowolnej liczby całkowitej $k \geq 1$, jeśli $a > b \geq 0$ oraz $b < F_{k+1}$, to wywołanie funkcji NWD(a,b) powoduje mniej niż k wywołań rekurencyjnych

Uwaga. Górne ograniczenie w twierdzeniu Lamego jest najlepszym możliwym oszacowaniem;

dla $k \geq 2$ wywołanie NWD(F_{k+1}, F_k) pociąga za sobą dokładnie $k-1$ wywołań rekurencyjnych

$$x^2 = x + 1$$

$$\phi = \frac{1+\sqrt{5}}{2} \qquad \phi^* = \frac{1-\sqrt{5}}{2}$$

ϕ – złota proporcja

$$F_k = \frac{\phi^k - \phi^{*k}}{\sqrt{5}}$$

$$F_k = \left\lfloor \frac{\phi^k}{\sqrt{5}} + \frac{1}{2} \right\rfloor$$

k-ta liczba Fibonacciego F_k jest równa wartości $\frac{\phi^k}{\sqrt{5}}$ zaokrąglonej do najbliższej liczby całkowitej

Stąd liczba wywołań rekurencyjnych wynosi $O(\lg b)$

Jeszcze dokładniej

Niech $c = \text{nwd}(a, b)$. Jeżeli $a > b \geq 0$ to obliczenie NWD(a, b) prowadzi do co najwyżej

$$1 + \log_\phi \left(\frac{b}{c} \right)$$

wywołań rekurencyjnych

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

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

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

Złożoność pesymistyczna

Dane wejściowe są w odwrotnym porządku

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

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

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

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 $A[j] \leftrightarrow A[j+1]$

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

Wersja z wartownikiem

SORT-BW (A, n)

$i = n$
while $i \neq 0$
  do $k = 0$
    for $j = 1$ **to** $i - 1$
      do if $A[j] > A[j+1]$
        then $A[j] \leftrightarrow A[j+1]$
          $k = j$
    $i = k$

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

Sortowanie przez scalanie

Schemat metody *dziel i zwyciężaj*

– **Dziel:** dzielimy tablicę na dwie podtablice po $n/2$ elementów każda

– **Zwyciężaj:** sortujemy otrzymane podtablice, używając rekurencyjnie sortowania przez scalanie

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

Krok Połącz

Dodatkowa procedura

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

W procedurze zakłada się, że podtablice $A[p..q]$ i $A[q+1..r]$ są posortowane

Procedura ta **scala** te podtablice w jedną posortowaną tablicę $A[p..r]$

```
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] = ∞
    C[m + 1] = ∞
    i = 1
    j = 1
    for k = p to r
        do if B[i] ≤ C[j]
                then A[k] = B[i]
                     i = i + 1
                else A[k] = C[j]
                     j = j + 1
```

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

gdzie $n = r - p + 1$ jest liczbą scalanych elementów

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

Procedura sortuje elementy w podtablicy $A[p..r]$

Jeżeli $p \geq r$, to podtablica zawiera co najwyżej jeden element więc jest już posortowana

W przeciwnym razie w kroku **dziel** znajdujemy indeks

$$q = (p + r) \text{ div } 2$$

który dzieli $A[p..r]$ na dwie podtablice: $A[p..q]$ and $A[q + 1..r]$

Krok **zwyciężaj** sortuje te podtablice rekurencyjnie wykorzystując procedurę SORT-SCAL

```
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 $(A, 1, n)$