NOTATKI I UZUPEŁNIENIA DO WYKŁADU 2

9 października 2016

Pewną modyfikacją procedury jest **procedura funkcyjna** zwana krótko **funkcją**. W tych procedurach *nazwa* pełni podwójną rolę, jako

– nazwa funkcji

– nazwa **obiektu** uzyskującego określoną wartość na skutek wywołania procedury

Boolean procedure P(WF)

integer procedure P(WF)

real procedure P(WF)

Przykład 14. Funkcja logiczna przyjmująca wartość **true** witw gdy dane dwa wektory $A[1..n]$ i $B[1..n]$ są równe, tj.

$$A[i] = B[i] \text{ dla } i = 1, \ldots, n$$

Boolean procedure RÓWNE (A, B, n)
1 $R =$ **true**
2 $i = 0$
3 **while** $i \neq n$ **and** R
4 **do** $i = i + 1$
5 $R = (A[i] == B[i])$
6 RÓWNE $= R$

Typeset by $\mathcal{AMS}$-TEX

Matematyczne pojęcie rekurencji

Przykład 15. Silnia

$$a_n = n \cdot a_{n-1} \qquad n \geq 1$$

z warunkiem początkowym $a_0 = 1$

Przykład 16. Ciąg Fibonacciego

$$a_n = a_{n-1} + a_{n-2} \qquad n \geq 2$$

z warunkami początkowymi $a_0 = 0$, $a_1 = 1$

Przykład 17. Wieże Hanoi

a_n - liczba ruchów potrzebnych do przeniesienia n krążków

$$a_n = 2a_{n-1} + 1 \qquad n \geq 2$$

z warunkiem początkowym $a_1 = 1$

Przykład 18. Współczynnik dwumianowy

$$\binom{n}{k} = \binom{n-1}{k-1} + \binom{n-1}{k} \qquad 0 < k < n$$

przy czym $\binom{n}{k} = 1$ dla $k = 0$ **lub** $k = n$

$b_{n,k}$ - wartość współczynnika $\binom{n}{k}$

$$b_{n,k} = b_{n-1,k-1} + b_{n-1,k} \qquad 0 < k < n$$

przy czym $b_{n,k} = 1$ dla $k = 0$ lub $k = n$

Przykład 19. NWD

Dla dowolnej nieujemnej liczby całkowitej a i dodatniej liczby całkowitej b zachodzi zależność

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

Warunek początkowy ?

Procedury rekurencyjne

Jednym z najbardziej użytecznych aspektów procedur jest rekurencja. Jest to zdolność procedury do wywołania samej siebie (jeden lub więcej razy) przy rozwiązywaniu podobnych problemów. Procedury rekurencyjne są kolejnym sposobem (po konstrukcjach iteracyjnych) odwzorowywania długotrwałych procesów na krótkie pseudokody.

Przykład 20. Silnia

$$a_n = n \cdot a_{n-1}, \qquad n \geq 1, \qquad a_0 = 1$$

integer procedure SILNIA (n)
1 **if** $n == 0$
2 **then** SILNIA$= 1$
3 **else** SILNIA$= n*$SILNIA$(n-1)$

Procedura iteracyjna

integer procedure SILNIA (n)
1 $s = 1$
2 **for** $i = 1$ **to** n
3 **do** $s = s * i$
4 SILNIA$= s$

Przykład 21. Ciąg Fibonacciego

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

integer procedure FIB (n)
1 **if** $n == 0$
2 **then** FIB= 0
3 **else if** $n == 1$
4 **then** FIB= 1
5 **else** FIB= FIB$(n-1)$+FIB$(n-2)$

Przykład 22. Największy wspólny dzielnik

Korzystamy ze wzoru rekurencyjnego na NWD: dla dowolnej nieujemnej liczby całkowitej a i **dodatniej** liczby całkowitej b zachodzi zależność

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

Niech a i b będą dowolnymi nieujemnymi liczbami całkowitymi

integer procedure NWD (a, b)
1 **if** $b == 0$
2 **then** NWD= a
3 **else** NWD= NWD$(b, a \bmod b)$

ZMIANA NOTACJI

FIB (n)
1 **if** $n == 0$
2 **then return** 0
3 **else if** $n == 1$
4 **then return** 1
5 **else return** FIB$(n-1)$+FIB$(n-2)$

```
NWD (a, b)
  if b == 0
    then return a
    else return NWD(b, a mod b)
```

return – zwraca wartość i przerywa pracę algorytmu, z wyjątkiem przekazania sterowania z powrotem do punktu wywołania w **procedurze wywołującej**

Techniki algorytmiczne

Dziel i zwyciężaj

Dziel: realizację problemu na dwie lub większą liczbę mniejszych części

Zwyciężaj: rozwiąż mniejsze podproblemy (przeważnie rekurencyjnie)

Połącz: rozwiązane podproblemy

Jest to przykład podejścia

zstępującego

top-down

```
FIB (n)
1  if n == 0
2   then return 0
3   else if n == 1
4         then return 1
5           else return FIB(n − 1)+FIB(n − 2)
```

Programowanie dynamiczne

Technika ta polega na tym, iż najpierw rozwiązywane są małe realizacje problemu, ich wyniki są przechowywane i w stosownym czasie algorytm może się do nich odwołać, zamiast liczyć je ponownie.

Jest ona zaliczana do podejścia **wstępującego** – *bottom-up*

Przykład 23. Ciąg Fibonaciego techniką programowania dynamicznego

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

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

Przykład 24. Współczynnik dwumianowy $\binom{n}{k}$ techniką programowania dynamicznego.
Wykorzystać tablicę dwuwymiarową $B[0..n, 0..k]$

$$b_{n,k} = b_{n-1,k-1} + b_{n-1,k} \qquad 0 < k < n$$

przy czym $b_{n,k} = 1$ dla $k = 0$ lub $k = n$

BIN (n, k)
for $i = 0$ **to** n
do for $j = 0$ **to** $\min(i, k)$
do if $(j == 0)$ **or** $(j == i)$
then $B[i, j] = 1$
else $B[i, j] = B[i-1, j-1] + B[i-1, j]$
return $B[n, k]$

Przykład 25. Wieże Hanoi

```
HANOI (n, X, Y, Z)
1  if n == 1
2     then X → Y
3     else HANOI (n − 1, X, Z, Y)
4          X → Y
5          HANOI (n − 1, Z, Y, X)
```

Wieże Hanoi iteracyjnie

1) przenieś najmniejszy krążek z palika, na którym właśnie spoczywa na palik następny zgodnie z ruchem wskazówek zegara
2) wykonaj jedyne możliwe przeniesienie nie dotyczące najmniejszego krążka

Uwaga. Algorytmy rekurencyjny i iteracyjny są równoważne, tj. oba dają te same sekwencje przeniesień krążków

Złożoność czasowa

Założenie: modelem obliczeń jest jednoprocesorowa maszyna o dostępie swobodnym do pamięci (RAM - *Random Access Machine*), a algorytmy są realizowane tak jak programy komputerowe (sekwencyjnie)

Założenie: złożoność czasowa musi być

własnością samego algorytmu

(niezależną od komputera czy języka programowania)

Operacjami dominujące

Za *jednostkę* złożoności czasowej przyjmuje się *wykonanie jednej* operacji dominującej

Asymptotyczna złożoność algorytmów

Dla dostatecznie dużych danych wejściowych liczymy jedynie rząd wielkości czasu działania algorytmu; wyrażamy ją za pomocą *notacji asymptotycznej*

Duże Θ

Dla danej funkcji $g(n)$ przez $\Theta(g(n))$ oznaczamy zbiór funkcji

$$\Theta(g(n)) = \{f(n) : \text{ istnieją dodatnie stałe } c_1, c_2, n_0$$

$$\text{takie, że } \; c_1 g(n) \leq f(n) \leq c_2 g(n) \; \text{ dla } n \geq n_0\}$$

Zapis

$$f(n) = \Theta(g(n))$$

$g(n)$ jest **asymptotycznie dokładnym oszacowaniem** dla $f(n)$

Duże O

Dla danej funkcji $g(n)$ przez $O(g(n))$ oznaczamy zbiór funkcji

$$O(g(n)) = \{f(n) : \text{ istnieją dodatnie stałe } c \text{ i } n_0$$

$$\text{takie, że } \; f(n) \leq cg(n) \; \text{ dla } n \geq n_0\}$$

Zapis

$$f(n) = O(g(n))$$

$g(n)$ jest **asymptotycznym ograniczeniem górnym** dla $f(n)$

Duże Ω

Dla danej funkcji $g(n)$ przez $\Omega(g(n))$ oznaczamy zbiór funkcji

$$\Omega(g(n)) = \{f(n) : \text{ istnieją dodatnie stałe } c \text{ i } n_0$$

$$\text{takie, że } \; cg(n) \leq f(n) \; \text{ dla } n \geq n_0\}$$

Zapis

$$f(n) = \Omega(g(n))$$

$g(n)$ jest **asymptotycznym ograniczeniem dolnym** dla $f(n)$

Przykład 26. Dany jest wektor $A[1..n]$.
Wyznaczyć najmniejszy indeks i składowej
o zadanej wartości x (**wyszukiwanie liniowe**)

ELEMENT-1 (A, n, x)
  $i = 1$
  while $(A[i] \neq x)$ **and** $(i \neq n)$
      do $i = i + 1$
  if $A[i] \neq x$
    then pisz komunikat
    else return i

Gdyby x był na pewno w tablicy

ELEMENT-2 (A, n, x)
  $i = 1$
  while $A[i] \neq x$
      do $i = i + 1$
  return i

  Rozważmy wektor $A[1..n + 1]$
  Wstawmy w $A[n + 1]$ element x (**wartownik**)

ELEMENT-3 (A, n, x)
  $A[n + 1] = x$
  $i = 1$
  while $A[i] \neq x$
      do $i = i + 1$
  if $i \leq n$
    then return i
    else pisz komunikat

Złożoność optymistyczna

Best-case time complexity $B(n)$

$$B(n) = O(1)$$

Złożoność pesymistyczna

Worst-case time complexity $W(n)$

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

Złożoność w średnim przypadku

Average-case time complexity $A(n)$

Zakładamy, że x występuje w tablicy oraz że prawdopodobieństwo tego, że x znajduje się na k-tej pozycji w tablicy, $1 \leq k \leq n$, wynosi $1/n$

$$A(n) = \sum_{k=1}^{n} \left(k \cdot \frac{1}{n} \right) = \frac{n+1}{2}$$

Czas działania algorytmu wynosi

$$\Theta(g(n))$$

wtedy i tylko wtedy, gdy
jego **pesymistyczny** czas działania wynosi

$$O(g(n))$$

a jego **optymistyczny** czas działania wynosi

$$\Omega(g(n))$$

Twierdzenie (*o rekurencji uniwersalnej*)
Niech $a \geq 1$ i $b > 1$ będą stałymi, niech $f(n)$ będzie pewną funkcją i niech $T(n)$ będzie zdefiniowane dla nieujemnych liczb całkowitych przez rekurencję

$$T(n) = aT(n/b) + f(n)$$

gdzie n/b interpretujemy jako $\lfloor n/b \rfloor$ lub $\lceil n/b \rceil$. Wtedy $T(n)$ może być ograniczona asymptotycznie w następujący sposób.

$$T(n) = aT(n/b) + f(n)$$

1. Jeśli $f(n) = O(n^{\log_b a - \epsilon})$ dla pewnej stałej $\epsilon > 0$, to
$$T(n) = \Theta(n^{\log_b a})$$
$$T(n) = aT(n/b) + f(n)$$

2. Jeśli $f(n) = \Theta(n^{\log_b a})$, to
$$T(n) = \Theta(n^{\log_b a} \lg n)$$
$$T(n) = aT(n/b) + f(n)$$

3. Jeśli $f(n) = \Omega(n^{\log_b a + \epsilon})$, dla pewnej stałej $\epsilon > 0$, to
$$T(n) = \Theta(f(n))$$

1. $f(n)$ musi być **wielomianowo mniejsza** niż $n^{\log_b a}$
2. $f(n)$ jest **tego samego rzędu** co $n^{\log_b a}$
3. $f(n)$ musi być **wielomianowo większa** niż $n^{\log_b a}$

Twierdzenia o rekurencji uniwersalnej nie można zastosować np. do rekurencji

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

$$a = 2,\ b = 2,\ f(n) = n \lg n$$

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

Funkcja $f(n)$ jest asymptotycznie większa od $n^{\log_b a}$ ale nie jest *wielomianowo* większa, bo

$$f(n)/n^{\log_b a} = (n \lg n)/n = \lg n$$

a $\lg n$ jest asymptotycznie mniejszy niż n^ϵ dla każdego $\epsilon > 0$. Rekurencja ta ,,wpada w lukę" między przypadki 2 i 3.