Analiza funkcji rekurencyjnych

Rekurencja -silnia

- Definicja rekurencyjna n!:

$$n! = \begin{cases} 1 & n = 0 \\ n * (n-1)! & n > 0 \end{cases}$$

Rekurencyjna definicja silni prowadzi do następującego algorytmu:

```
unsigned int Factorial (unsigned int n)
{
(1) if (n == 0)
(2)       return 1;
    else
(3)       return n * Factorial (n - 1);
}
```

Analiza złożoności:

instr	n=0	n>0
1	*O(1)*	*O(1)*
2	*O(1)*	-
3	-	*T(n-1)+O(1)*
razem	*O(1)*	*T(n-1)+O(1)*

Z tabeli wynika, że złożoność czasowa rekurencyjnego algorytmu *Factorial* dana jest rekurencyjnym wzorem:

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

Jak rozwiązać takie równanie?

Pominiemy O(.), rozwiążemy równanie i wstawimy O(.).

Rozwiązujemy więc równanie:

$$T(n) = \begin{cases} 1 & n = 0 \\ T(n-1) + 1 & n > 0 \end{cases}$$

Jest to tak zwane równanie rekurencyjne.

Najprostszą metodą rozwiązywania równań rekurencyjnych jest metoda powtórzonych podstawień (nazywana też **rozwinięciem równania do sumy**):

1. Rozpisujemy równania jawnie dla wszystkich *n*.
   $T(n) = T(n-1) + 1$
   $T(n-1) = T(n-2) + 1$
   .
   .
   .
   $T(1) = T(0) + 1$
   $T(0) = 1$
2. Po podstawieniach otrzymujemy:
   $$T(n) = T(n-1) + 1 = T(n-2) + 1 + 1 = T(0) + n = 1 + n$$
   czyli rekurencyjny algorytm obliczania silni jest klasy *O(n)*

Zad.1. Zaimplementuj funkcję rekurencyjną obliczającą x^n. Podaj równanie rekurencyjne określające jej złożoność i oblicz złożoność.

Zad.2. Zaimplementuj funkcję rekurencyjną obliczającą sumę ciągu arytmetycznego S(int n). (S(n)=1+2+…+n), podaj równanie rekurencyjne określające jej złożoność i oblicz złożoność.

Zad.3. Zaimplementuj funkcję rekurencyjną obliczającą następującą sumę: Sum(int n). ($Sum(n)=1+2^2+2^3\ldots+n^n$),zakałdając, że mamy funkcję p(n) obliczającą n^n rzędu O(n). Podaj równanie rekurencyjne określające jej złożoność i oblicz złożoność.

Zad.4. Mamy następujące równanie rekurencyjne:

$$T(n) = \begin{cases} 0 & n = 1 \\ T(\lfloor n/2 \rfloor) + c & n > 1 \end{cases}$$

(równanie to otrzymujemy jako równanie złożoności wtedy, kiedy problem rozmiaru *n* sprowadza się do problemu o połowę mniejszego).
Trudno jest w tym przypadku zastosować rozwinięcie równania do sumy.

Zastosujemy zmianę zmiennych: podstawiamy $n = 2^k$

$$T(2^k) = T(2^k / 2) + c = T(2^{k-1}) + c$$

Teraz możemy zastosować metodę rozwinięcia równania do sumy:

$$T(2^k) = T(2^{k-1}) + c = T(2^{k-2}) + c + c = T(2^0) + kc = kc = c \log n$$

Stąd wynika, że T(n)=O(log*n*)