# Rekurencja

26 października 2015

## 1 Rekurencja

Rekurancja polega na rozbiciu problemu na coraz mniejsze podproblemu, aż do momentu napotaknia problemu który może być rozwiązany w sposób trywialny. Zazwyczaj w takim wypadku korzysta się z funkcji wywołującej samą siebie. Rozwiązania rekurencyjne okazują mogą być prostsze do zaimplementowania od tych oferowanych przez inne metody. Przykład obliczanie silni: Algorytmy korzystające z rekurencji muszą spełniać trzy prawa:

**Algorithm 1** Silnia

```
function FACTORIAL(n)
    if n==1 then
        return 1
    end if
    return n*FACTORIAL(n-1)
```

1. Algorytm musi mieć przypadek podstawowy (dla silni przypadek taki ma miejsce gdy n=1)
2. Algorytm musi zmieniać swój stan i poruszać się w kierunku przypadku podstawowego (dla silni w każdej instancji następuje przemnożenie przez n)
3. Algorytm musi wywoływać samego siebie (dla silni w każdej instancji następuje wywołanie algorytmy dla n-1)

## 2 Labirynt

Rekurencją monżna posłużyć się również w celu odnalezienia wyjąścia z labiryntu. Załóżmy że labirynt jest zadany przez 2-wymiarową tablicę (x,y). Poruszający się po labiryncie ma 4 możliwości:

- pójść w górę (x+1)
- póść w dół (x-1)
- pójść w lewo (y+1)
- póść w prawo (y-1)

Poruszając się po labiryncie nie można wchodzić na przeszkody ani wychodzić poza jego obszar. Rekurencyjny algorytm znajdujący wyjście można schematycznie zapisać następująco (zobacz pseudokod algorytmu 2):

## 3 Wieża Hanoi

Problem polega na przeniesieniu n klocków ze stosu początkowego na stos końcowy. Przy przenoszeniu klocków należy pamiętać o tym, że klocki większe nie mogą być układane na mniejszych. Do wykonania zadania można posłużyć się stosem pomocniczym. Problem należy rozwiązać rekurencyjnie. Jeżeli wiemy w jaki sposób przenieś n-1 elementowy stos, wiemy również jak poradzić sobie z n elementowym stosem. W tym celu wystarczy:

- przenieść n-1 elementowy stos na stos pomocniczą (jako stos pomocniczy wykorzystując stos końcowy)
- przenieść pozostały element na stos końcowy

**Algorithm 2** Ścieżka

```
function ŚCIEŻKA(x,y)
    if x,y poza obszarem labiryntu then return false
    end if
    if x,y to współrzędne przeszkody lub ścieżka była eksplorowana then return false
    end if
    if x,y to współrzędne wyjścia then return true
    end if
    Oznacz pole x,y jako zeksplorowane
    if Ścieżka(x+1,y) == true then droga[x,y] = true
     return true
    end if
    if Ścieżka(x-1,y) == true then droga[x,y] = true
     return true
    end if
    if Ścieżka(x,y-1) == true then droga[x,y] = true
     return true
    end if
    if Ścieżka(x,y+1) == true then droga[x,y] = true
     return true
    end ifreturn False
```

- przenieść n-1 elementowy stos na stos końcowy (jako stos pmocniczy wykorzystując stos początkową)

Wejściem do programu jest lista n elementowa [1,...,n] oznaczająca klocki do przeniesienia. W trakcie wykonywania operacji należy wyświetlać zawartości wszystkich trzech stosów.
Zadania

1. Zaimplementować rekurencyjny algorytm odnajdujący drogę w labiryncie. Wejściem powinna być tablica dwuwymiarowa, określająca przeszkody w labiryncie. Można przyjąć, że wyjście z labiryntu zawsze znajduje się w komórce [0,0]. Na koniec program powinien wyświetlić tablicę zawierającą zbadaną przez algorytm scieżkę w labiryncie. Za poprawną implementację można otrzymać 5 punktów.

2. Zaimplementować algorytm rozwiązujący problem wieży Hanoi dla zadanej przez użytkownika wielkości początkowej stosu. W kolejnych krokach działania algorytmu należy wyświetlać na ekranie stan każdego ze stosów. Za poprawnie napisane zadanie otrzymać można 5 punktów.