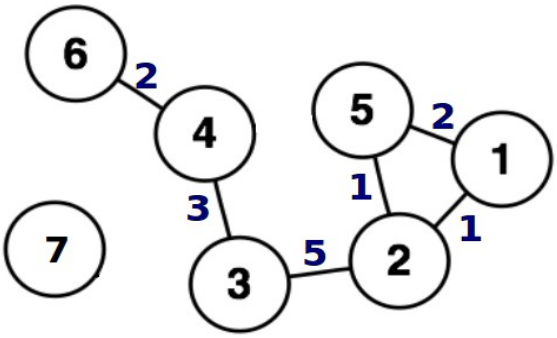
Języki Programowania, Laboratorium 11, 4.01.2012

Wyszukiwanie ścieżki w grafie. Macierz sąsiedztwa. Algorytm Djikstry. Czytanie pseudokodu.

W teorii grafów **macierz sąsiedztwa** grafu *G* jest kwadratową macierzą w której a_{ij} oznacza wagę krawędzi pomiędzy wierzchołkami *i* i *j*. Dla przykładu – graf z 7 wierzchołkami:



$$A = \begin{pmatrix} 0 & 1 & 0 & 0 & 2 & 0 & 0 \\ 1 & 0 & 5 & 0 & 1 & 0 & 0 \\ 0 & 5 & 0 & 3 & 0 & 0 & 0 \\ 0 & 0 & 3 & 0 & 0 & 2 & 0 \\ 2 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

1. Należy stworzyć klasę graf używając dwuwymiarowej tablicy (macierz sąsiedztwa) do przechowywania informacji o jej krawędziach. Klasa powinna mieć następujące metody:

`Graf(int max)` – konstruktor ; `max` – maksymalna ilość wierzchołków w danym grafie. W przypadku osiągnięcia tej liczby dodanie nowego wierzchołka nie będzie możliwe.

`void init()` - zeruje wszystkie pola macierzy sąsiedztwa

`void add()` - dodanie wierzchołka

`void remove(int r)` - usunięcie wierzchołka o numerze r

`void display()` - wyświetlenie grafu (macierzy sąsiedztwa)

`void add_edge(int n, int m, int w=1)`- dodanie krawędzi pomiędzy wierzchołkami o numerach m i n (oraz wagą w)

`void remove_edge()` - usunięcie krawędzi

W funkcji main stworzyć graf odpowiadający podanemu powyżej w przykładzie.

Dynamiczne tworzenie dwuwymiarowych tablic w C++:

```
int **tab = new int *[5];
for (int i = 0; i < 5; ++i)
   tab[i] = new int [10];
```

W ten sposób stworzono tablicę dwuwymiarową którą statycznie zadeklarowalibyśmy jako:

```
int tab[5][10];
```

2. Należy zaimplementować algorytm djikstry do wyszukiwania najkrótszej ścieżki w grafie jako metodę klasy graf na podstawie podanego na następnej stronie pesudokodu.

`void dijkstra(int source, int target)` - wyszukiwanie najkrótszej ścieżki od source do target

Oraz przetestować go dla stworzonego wcześniej grafu, oraz ścieżek: (1,3), (6,7), (6, 5).

3. Rozszerzyć klasę graf w ten sposób, by maksymalna liczba wierzchołków w grafie zmieniała się, jeśli będziemy chcieli dodać kolejny wierzchołek, przekraczający tę liczbę.

```
function Dijkstra(Graph, source, target):
      for each vertex v in Graph: // Inicjalizacje, dla każdego wierzchołka
        distance[v] := infinity ; // Odległość od source do v (nieznana, zakładamy nieskończoność)
        previous[v] := -1 ;        // Poprzedni wierzchołek na najkrótszej ścieżce
        VISITED[v] := 0;          // Żaden z wierzchołków nie był odwiedzony
      end for ;
      distance[source] := 0 ;
      if source = target          // Jeśli source jest równe target - ten sam wierzchołek
            print "The same vertex"
            exit

      int tmp := infinity;
      int visited_nodes := 0;
      int current = source;

      while visited_nodes < Graph size
            for each node x of Graph
                  if node x was not visited and there exist edge between x and current
                        if distance[current] + distance between current and x < distance[x]
                              distance[x] := distance[current] + distance between current and x
                              previous[x] := current
            end for

            VISITED[current] := 1                  // Wierzchołek current odwiedzono
            visited_nodes := visited_nodes + 1

            for each vertex v in Graph
                  if vertex v was not visited and distance[v] < tmp
                        tmp := distance[v]
                        current := v
            end for

            tmp := infinity

            if current = target
                  print "Path found"
                  int u := target;
                  while previous[u] != -1
                        print u
                        u := previous[u]
                  print source

      print "Path not found"

 end Djikstra
```