INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES RENNES

C language 2

Lab 12
S5 EII

2020-2021

Contents

INSA

LAB 12

Graphs : Dijkstra's algorithm

Objectives : (*2h*)
- Implement functionalities for a weighted directed graph implemented using linked lists ;
- Implement Dijkstra's algorithm to find the shortest path

Requirement :
- read the course document on graphs ;
- read the whole practical subject ;
- write the Makefile for the project before the practical session.

12.1 Overview of the practical works on graphs

You will first implement a graph module in the first session. In the second session, you will use its functionalities to solve the Dijkstra's algorithm to find the shortest path between two vertices.

12.2 Weighted graph

We will study a weighted directed graph. The value stored by the edges represents the distance between two vertices.

A vertex is such as :

```
struct Vertex{
    char label;              /*!< the vertex label*/
    EdgeList connect;        /*!< the successor vertices linked to this vertex*/
    struct Vertex *next_v;   /*!< the address of the next element in the list*/
    int already_visited;     /*!< a flag to indicate whether the vertex has
        been processed or not in the Dijkstra's algorithm.*/
    int dist_to_origin;      /*!< the distance from the origin vertex in the
        Dijkstra's algorithm. */
    EdgeList path;           /*!< the path followed to get the minimal distance
        from the origin vertex in the Dijkstra's algorithm. */
};
```

where :
- *label* is the value of the vertex.
- *connect* contains a list of edges to describe the vertex connections in the graph ;
- *next_v* is a link to another vertex of the graph ; (nothing to do with the vertex connections, it only helps for the graph vertices list) ;
- Some members are required by the Dijkstra's algorithm as seen in course : *already_visited*, *dist_to_origin*, *path*.

An edge represents a successor of a vertex, as defined in the course :

```
struct Edge{
    Vertex *v;              /*!< the successor vertex*/
    int dist;               /*!< The distance between the two vertices*/
    struct Edge *next_e;    /*!< the address of the next element in the list*/
};
```

where :

- *v* is the destination vertex of the edge, the source being the vertex containing the concerned edge list ;
- *dist* is the value of the edge ;
- *next_e* is a link to the next edge of the vertex whose belongs the edge list.

Since the main objectives of this practical work is to focus on graphs, the two kind of lists (vertices and edges) implementations are provided for the OS used in these labs.

As seen in the course, a graph is a list of vertices :

```
typedef VertexList Graph;
```

The following functionalities are provided :

```
void initGraph(Graph *g);
void deleteGraph(Graph *g);
void printGraph(Graph *g);
int addVertexGraph(Graph *g, char val);
Vertex * findVertexGraph(Graph * g, char val);
int deleteVertexGraph(Graph * g, char val);
int addEdgeGraph(Graph *g, char val1, char val2, int d);
Edge * findEdgeGraph(Graph * g, char val1, char val2);
int deleteEdgeGraph(Graph * g, char val1, char val2);
```

Question 1 : write a Makefile to build a project with the provided files.

Question 2 : create the following functionality :

```
void writeGraphViz(Graph *g, char * filename);
```

It should write a file in the graphviz format with the data corresponding to the graph given in parameter. You will find in Figure 12.1 an example for a directed graph :

To visualize such a file, you can use *xdot* viewer. If it is not yet installed, type :

```
$ sudo apt-get install graphviz xdot
```

```
digraph{
  a -> c;
  a -> b;
  b -> c;
  d;
}
```

gives :

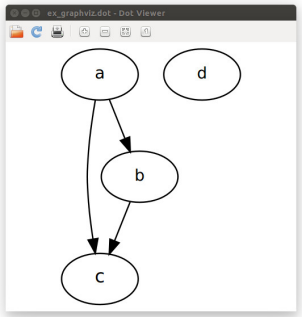

FIGURE 12.1 – DOT format for a directed graph

12.3 Dijkstra's algorithm

Question 1 : using these functionalities, implement the Dijkstra's algorithm.

As seen during the lessons, the Dijkstra's algorithm is such as :

Initialization : set all vertices as unvisited, with a distance to v_1 set to infinity
Then proceed iteratively :

1. Search for the unvisited vertex v_i with the smallest distance to v_1 such as $dist(v_1, v_i) < dist(v_1, v_2)$
2. If there is no candidate, end of algorithm
3. Candidate v_i found :
   (a) Mark v_i as visited
   (b) For each neighbour v_j of v_i, update its distance to v_1 :
   $dist(v_1, v_j) = \min(dist(v_1, v_j), dist(v_1, v_i) + dist(v_i, v_j))$

It is strongly recommended to split up the problem in functions, one for each part of the algorithm : initialization, finding the next vertex, updating the distances of the neighbours of the chosen vertex.

12.4 (Optionnal) Using adjacency matrix

Solve the same problem using the 2D array (matrix of adjacency) representation of a graph.

INSA Rennes
20 Avenue des Buttes de Coësmes
CS 70839
35708 Rennes Cedex 7

Tél. +33 (0) 2 23 23 82 00
Fax +33 (0) 2 23 23 83 96

www.insa-rennes.fr

INSA

UNIVERSITÉ EUROPÉENNE DE BRETAGNE

Commission des Titres d'Ingénieur

MINISTÈRE DE L'ENSEIGNEMENT SUPÉRIEUR ET DE LA RECHERCHE