Project

OPTIMAL ROUTING

IMPORTANT: Read the entire document carefully before you start working. This will help you avoid unpleasant surprises.

The most well-known network problems are the information dissemination problem and the deadlock detection problem. For example, in a network of robots where resources are very limited, we want to send data from one robot to another with minimal cost and we want to be sure that the data will be delivered without falling into an infinite loop.

To resolve a optimal routing probleme, we will use a Floyd-Warshall algorithm which allows one to compute minimum value paths from any vertex to any other vertex in an oriented and valued graph. This algorithm is based on the Warshall algorithm of computation of the transitive closure of a graph. The computation of transitive closure is adapted in order to keep, among all the paths connecting 2 vertices, one of those having the smallest value.

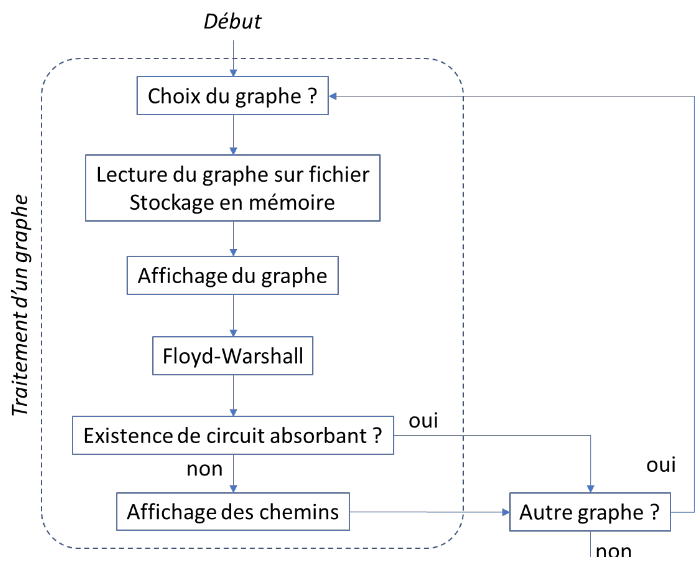
The Floyd-Warshall algorithm is not discussed either in exercises or in lectures. This project therefore requires some research work on your part.

GENERAL CONDITIONS

The project is to be carried out in teams by 2 persons.
You will need to use one of the following programming languages: C, C++, Java, or Python.
You have to send the result of your work to your teacher beforehand (the date will be sent late). Test graphs, on which your program will be tested, will be sent to you some time before the presentation. You do not need those test graphs when developing the program: the program should work on any graph.

WORK TO DO

Illustration :



Main loop:

Your program must be able to execute the requested operations on a series of graphs, at the user's choice. During the presentation, it should not be necessary to restart your program to test the next graph.

Processing of a graph:

First, the user tells you which graph should be analyzed. The graphs will be identified by their sequential numbers.

The graph is then loaded into memory according to its description contained in a text file. See Appendix.

The "read" graph is stored in data structures. From this point on, your program must never re-access the text file: only the memory representation is used.

The choice of representing the graph in memory as a string of characters mimicking the lines of the text file will not be efficient if your program has to process large graphs. This is true whatever you dress it in, including tables, vectors, sets, ... The triplets suggested for a text representation of a graph are not a good choice for memory storage. **Your choice of a more efficient data structure will play a role in your grade.**

The graph is then displayed on the screen as a matrix: an adjacency matrix plus a matrix of values, or both combined.

Pay close attention to the legibility of this display, especially the alignment of the columns and the identification of the vertices (row and column titles).

The Floyd-Warshall algorithm is then executed. The intermediate values of the algorithm data (matrices usually named 'L' and 'P') should be displayed.

When you come to the results of the algorithm, your program should first of all say if the graph contains at least one absorbent circuit.

If no absorbent circuit is present, the last operation consists of displaying the shortest paths. You can display them all, or provide a user interface loop:

```
Path ?
If Yes then
    Initial vertex?
    Final vertex?
    Displaying the path
    Redo
If No then stop
```

GRAPHS TO CONSIDER

Your program must be able to correctly process any graph that meets the criteria below:

- The graph is oriented
- The graph is valued by arbitrary integer numbers (negative integers OK, zero value OK)
- The vertices are labeled by integers from 0 to the number of vertices minus one
- There is at most one edge going from vertex x to vertex y

Your program should not impose any limits on the number of vertices, values of edges, or number of edges.

Test graphs

Test graphs in graphic form will be provided to you before the presentation, early enough to allow you to code them into .txt files in the format of your choice. These files must be prepared in advance, and they will be a part of the submission package. They must be available on your computer at the time of the presentation.

To test your program, don't wait for these graphs! It would then be too late.

During your work, don't hesitate to use all the graphs seen in lectures and exercises, and many more. The more testing you do, the more you can be sure that your program works properly.

Appendix – Example of a file representing a graph

Careful : this appendix contains only an ***example*** file structure. You have the right to decide on your own structure if you respect the following conditions:

- the file structure must be simple, easily understandable ;
- the graph represented in the file must directly modifiable in the file, in an extremely simple way (for example if you wish to add or delete an edge, or to change the value of an edge).

Your file can, *for example,* have the following structure:

Line 1	Number of vertices
Line 2	Number of edges
Lines 3 to (3 + the number of edges)	Initial vertex followed by final vertex followed by the edge value

With that model, if the file contains the following text:

```
4
5
3 1 25
1 0 12
2 0 -5
0 1 0
2 1 7
```

The graph will have 4 vertices numbered from 0 to 3 (the numbering must be contiguous) and 5 edges (3,1), (1,0), (2,0), (0,1), (2,1) whose values are, respectively, 25, 12, -5, 0 and 7.