EFREI 2019/2020
L3INT

Graph Theory
Mini-project

Goal: Scheduling

WARNING
1. The project is divided into two distinct parts.
2. If you do just Part I, it will not allow you to achieve the maximum score of 20/20. The total score will probably be broken down as follows: Part I = 12, Part II = 8.

Work to do :

Work will be done in teams.

Groupe Int1 has 32 students. This means that you'll form 10 teams of 3 and one team of 2
Groupe Int2 has 29 students. This means you'll form 9 teams of 3 and one team of 2.
The constitution of the teams: to be submitted to me by March 27th.

No changes will be allowed after this date. In the absence of a team composition provided by the delegates of each TD group, I'll have the right to decide on the constitution of the teams myself, without any possibility of modifying my decision.

Programming language

You can choose between: C, C++, Python, Java.
However, the language chosen must be sufficiently mastered by all members of the same team so that everyone can participate. During the defense, your teacher may put any question to any member of the team.
Your program must be compileable / executable by me, on my PC (if I wish to do so). I have CodeBlocks 13.12, Eclipse Java 2018-09, and Anaconda Spider 3.3.3 (not that I have really used the latter two for programming).

Project defense

Presentation + demonstration of your program (precise conditions will be given later) + questions/answers.
In case the containment remains in place at the date of the presentation, it will be done on Teams or other teleconferencing tool according to modalities that we have yet to establish.

The duration of presentations is limited. The time schedule will be very tight. I will not be able to allow you more time than scheduled.
You are therefore strongly advised to be absolutely ready at the time of the beginning of your presentation, which implies (in case a physical presentation takes place):

- waiting at the door of the room and come in as soon as it's your turn;
- having prepared your computer, on which you'll have all your programs as well as the text graph files;
- having checked that your computer's battery is fully charged;
- having started it up and then put it into sleep mode;
- having a backup computer on which you've also checked that your program is working properly, just in case...

Every year, there are students who think they'll never have a problem. And every year, there are those who do have a problem. Result: they are nervous and have less time for their defense. A pity…

Test graphs

Test graphs will be given to you in graphical form, sufficiently early to allow you to recode them into .txt files in the format of your choice. These files must be prepared in advance, and they will be part of what you'll send me. They must be available on your computer at the time of the presentation.

Don't wait for these graphs to start testing your program! It would then be too late.
Don't hesitate to use and test all the graphs you have seen in lectures and exercise sessions and elsewhere. The more testing you do, the more certain you can be that your program works properly.

Sending in your work

All teams must submit their work, by email, by the same date. Detailed instructions will be given to you at a later date.

The content of what you must send in:

- Source code: Only code files that you typed yourself (absolutely no file produced by the software during compilation or execution!), well commented.
- All the .txt files of the test graphs (yes, despite the fact that it's me who'll give you the test graphs), in the same directory as the code.
- A ppt or pdf of the presentation.
- The execution traces in the form of a .txt file. You will have to run your program on all of the test graphs and provide the corresponding execution traces.

A graph that is not tested, or for which the execution traces are not provided, will be considered as a graph on which your program does not run correctly.

Any file you use (and pass on to your teacher) must be prefixed with your team number: for example, if you have a "main" file and you use C++, and you are in team Int2-3, this file must have the name Int2-3-main.cpp (or Int2-3_main.cpp)

Part I: reading and displaying a graph. Detecting a cycle. Computing the rank.

Graphs to consider

Your program must be capable to deal correctly with any graph described below:

- The graph is directed
- Vertices are numbered starting from 0
- There may not be any break in the numbering (for example, the vertices of a graph consisting of 15 vertices will be numbered from 0 to 14).
- The graph is weighted:
  1 numeric value per edge; no restriction on the values: they can be negative, zero, or positive
  At most one edge from any vertex 'x' to vertex 'y'
- There is no limit on the number of vertices. In particular, your program must not contain any hard definition of the number of vertices or edges. It must adapt to what it reads in the graph file.

The program

Set up a program that performs the following actions :

1. Reading a given from a text (.txt) file and storing it in memory (See Annex 1).
2. Displaying of the graph in matrix form (adjacency matrix + value matrix). Caution: this display must be done from the memory content, and not directly when reading the file.
3. Detecting the presence or absence of a cycle in the graph (use the method of your choice).
4. If the graph does not contain a cycle, calculating the rank of each vertex (use the method of your choice).

Your program must allow the user to test several graphs without restarting.
If your program stops after each graph and has to be relaunched, it is not a good solution, and it will result in a penalty.

The global structure of the program (Part I) can be shown by the following pseudocode:

```
begin
    while the user wants to test a new graph do
        choose the text file by its number
        read the corresponding file and save the graph in memory
        display the adjacency matrix and the value matrix from memory
        detect if there is a cycle
        if there is no cycle then
            calculate the ranks of the vertices and display them
        endif
    done
end
```

The 4 steps must be implemented individually.
Of course, it is possible to combine the circuit detection and the rank calculation in one algorithm. But the goal of the Project is to make you learn as many key algorithms of graph theory as possible, and therefore this solution is forbidden.

Execution traces

Each of the 4 steps must produce traces of the algorithm's execution. Here's an example:

Step	Exemple of trace (those are just examples ! You can do what you want provided that one can, when looking at your traces, easily and quickly understand the functioning of your algorithm).

1	*Displaying, for each line of the input file, what has been read. For example, if you adopt the structure shown in Annex 1 and the graph given there as illustration, you'll have to produce something like that:*
	`* Reading the graph from file`
	`4 vertices`
	`5 edges`
	`3 -> 1 = 25`
	`1 -> 0 = 12`
	`2 -> 0 = -5`
	`0 -> 1 = 0`
	`2 -> 1 = 7`

2	*Keeping the example of Annex 1:*
	`* Representation of the graph by matrices`
	`Adjacency Matrix`
	`  0 1 2 3  or  0 1 2 3`
	` 0 F V F F  0 0 1 0 0`
	` 1 V F F F  1 1 0 0 0`
	` 2 V V F F  2 1 1 0 0`
	` 3 F V F F  3 0 1 0 0`
	
	`Matrix of Values`
	`  0 1 2 3`
	`0 * 0 * *`
	`1 12 * * *`
	`2 -5 7 * *`
	`3 * 25 * *`

3	*For example, using the method of eliminating the entry points:*
	`* Detecting the cycle`
	`* Method of elimination of entry points`
	`2 3`
	`Elimination of entry points:`
	`Remaining vertices:`
	`0 1`
	`Entry points:`
	`None`
	`The graph contains at least one cycle.`

4	*For example, using the graph of Annex 1 but removing one edge (to get rid of the cycle)*
	4
	4
	3 1 25
	1 0 12
	2 0 -5
	2 1 7
	`* Calculating the ranks`
	`* Method of elimination of entry points`
	`Current rank = 0`
	`Entry points:`
	`2 3`
	`Current rank = 1`
	`Entry points:`
	`1`
	`Current rank = 2`
	`Entry points:`

	0 Graphe is empty Computed ranks: Vertex 0 1 2 3 Rank 2 1 0 0

Part II: Scheduling (computing the calendars)

In Part I we deal with any graphs.
In Part II we deals with graphs suitable for scheduling (see properties in item 5).

The realization of part II is a continuation of what has been done in part I. The code of part I is used by the functions to be developed in part II.

Additional functions to develop

5. Check if the graph is a "correct" scheduling graph, i.e. such that it respects the properties you've seen in lectures and in exercise sessions:
   - a single entry point,
   - a single exit point,
   - no cycle,
   - same weights for all outgoing edges of a vertex,
   - outgoing edges of the entry vertex have zero,
   - no negative arcs.
6. If the answer to point 5 is "yes", calculate the earliest and latest dates and the margins.
   To calculate the latest dates, you must assume that the latest project completion date is equal to the earliest project completion date.

The global structure of your program, which is now modified as compares to Part I, can be summed up in the following pseudocode:

```
begin
    while the user wants to test a new graph do
        choose the text file by its number
        read the corresponding file and save the graph in memory
        display the adjacency matrix and the value matrix from memory
        detect if there is a cycle
        calculate the shortest paths
        if there is no cycle then
            calculate the ranks of the vertices and display them
            if the graph is a scheduling graph then
                compute and display the calendars
            endif
        endif
    done
end
```

Note that in the test "if the graph is a scheduling graph", there is no need to re-test the presence or absence of a cycle. Only additional conditions need to be tested.

Annex 1: Structure of the graph file

Important: This Annex contains just one example of the structure. You may decide that you'll use a different structure, provided that:

- It is simple and easy to understand,
- It must be very simple to modify the graph that the file represents (for example, to add an edge, to remove an edge, to change the weight of an edge).

For example, the file may be like that:

Line 1	number of vertices
Line 2	number of edges
Line 3 to (3+the number of vertices)	initial end of an edge, final end, weight

With that file 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
```

This corresponds to a graph containing 4 vertices numbered from 0 to 3 (an uninterrupted numbering) and 5 edges (3,1), (1,0), (2,0), (0,1), (2,1) whose respective weights are 25, 12, -5, 0, and 7.

Communication

The email address to use in any communication and sending in files is **velikson.efrei@gmail.com.**
All emails must have as subject prefix the string TG-PRJ-<team number>, possibly followed by the subject of your email. For example, when submitting the work, team Int1-5 may send an email with the subject line "TG-PRJ-Int1-5 Final Submission".

These instructions are very important and even partial violation of them may result in penalties.
I have automated the mail client so that this way the mails get correctly. Any failure on your part can lead to an incorrect treatment of your work (this can go as far as spamming your emails...), with all the consequences that this may entail.

Programming languages

Choose between: C, C++, Python, Java

This text has been translated from French by me with very slight changes (like replacing "les enseignants" by "me"), but without spending much time on re-reading. If you see errors, tell me. If you have other questions, write to me as well. The French original, elaborated by M Barbot and me, is available on the Moodle space of SM601.
Good luck
Boris Velikson