Université d'Ottawa
Faculté de génie

École de science informatique
et de génie électrique

uOttawa

L'Université canadienne
Canada's university

University of Ottawa
Faculty of Engineering

School of Electrical Engineering
and Computer Science

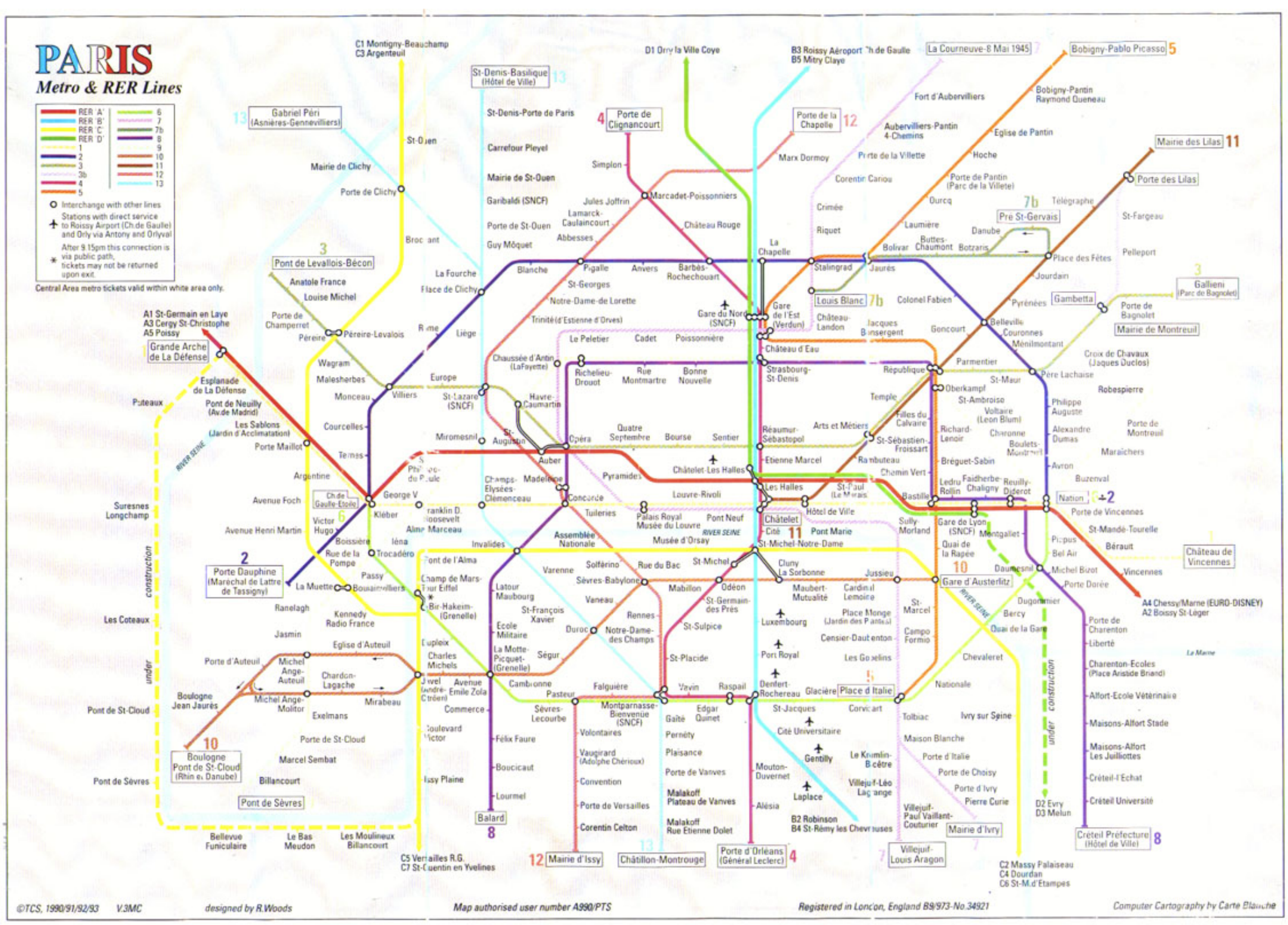
Programming Assignment P2 (10%)

CSI2110, Fall 2023, Professor Lucia Moura

Due date: Monday November 27, 11:59PM (from 1 min to 24hs late receives 30% late penalty)

Paris Metro: Exploring the Paris Subway Network

The objective of this assignment is for you to get practice applying several graph algorithms in a relatively complex graph. For this, we will use the subway (metro) of the City of Paris. In this handout the word "lines" refer to subway lines, which in the picture below are "lines" of different colours (note that lines can be more complex than simple paths in a graph).



The information for you to build the graph for Paris subway network is contained in the textfile metro.txt. This graph is **directed** in order to take into account for some cases where the link is only one-way. This file is organized as follows:

- The file starts by specifying two integers: the number N of vertices and the number M of edges in the graph.
- The next N lines in the file specify N vertices (stations). Each vertex has a unique number followed by a station name. Each vertex corresponds to a point in a subway line corresponding to a particular station in that line. Note that different vertices can be followed by the same station name when several subway lines pass by the same "physical station" (e.g. Bastille). For the purpose of this handout each vertex corresponds to a station, and stations with different numbers and the same name are considered as different "stations" (e.g. stations 16, 17, 18). Examples of vertices:
    - 0016 Bastille
    - 0017 Bastille
    - 0018 Bastille
    - 0119 Gare de Lyon
    - 0135 Hôtel de Ville
    - 0331 Saint-Paul, Le Marais
- The symbol $ denotes the end of the vertex list.
- The rest of the file lists the edges of our graph, i.e. the existing connections between the subway stations. The format is v1 v2 w, where v1 and v2 are vertex numbers and w is a number representing the weight of the edge.

  If the weight is positive it indicates the time required (in seconds) to go from one station to the other station as adjacent stations on a subway line. For example the following edges connect adjacent stations in the light yellow line from right to left in the center of the map, connecting 0119 Gare de Lyon -> 0016 Bastille -> 0331 Saint-Paul, Le Marais -> 0135 Hôtel de Ville, taking 98 secs, 62 secs and 65 secs, respectively:
    - 119 16 98
    - 16 331 62
    - 331 135 65

  If the weight of an edge is -1, this edge indicates that it is possible to walk in order to switch from a station in one line to an station in another line. The time estimated when traversing an edge of weight -1 is a constant specified in your program for the walking time, which you must set as **90 secs**. For example, the following links represent that we can walk among the three vertices in Bastille (16, 17, 18):
    - 16 18 -1
    - 16 17 -1
    - 17 18 -1
    - 17 16 -1
    - 18 16 -1
    - 18 17 -1

Your tasks

1. You must read the file and create a graph using the representation of your choice.

2. Using this graph, your program needs to allow to automatically:
   i) Identify all the stations belonging to the same line of a given station.
   ii) Find the shortest path between any two stations, i.e. the path taking the minimum total time. Print all the stations of the path in order, and print the total travel time.
   iii) Find the shortest path between two stations when we have the information that one given line is not functioning (the line is identified by one of its endpoints). The same type of information will be printed as in ii).
   iv) Bonus question (optional): this is a harder question, dependent on the previous questions and described separately in the last page of this handout.

   Note: when we mention "station" in items above we mean the station **number**.

Your program:

You should create the following classes:

class `Graph`: This class stores the vertices and edges of a graph and implements the required basic graph methods.

class `ParisMetro`: This class deals with the operations for dealing with the Paris subway network. It will have a member object of the graph class to store the subway graph, a static method `readMetro(String fileName)` to read the input file, any auxiliary methods implementing algorithms you need to answer the assignment questions and a main method to run the program with the input arguments specified next.

You have freedom in the way you design these classes and you may use other classes if you wish.

In order to run your program from the command line, the call should be as follows:

```
java ParisMetro N1 N2 N3
```

with `N1`, `N2` and `N3` being numbers identifying subway stations.

- If only `N1` is specified the program must answer question 2-i) where `N1` is the given station identifying the line to be printed.
- If `N1` and `N2` are specified the program must answer question 2-ii) where `N1` is the departure station and `N2` is the arrival station.
- If `N1, N2` and `N3` are specified the program must answer question 2-iii) where `N1` is the departure station, `N2` is the arrival station and `N3` is the endpoint of a broken line (line that is not functioning).
- If no parameter is specified the program must answer 2-iv) the bonus question (optional).

What to submit

1. **Java code:** Submit the classes to run your program, following the specifications in the previous pages. This should contain `ParisMetro.java`, `Graph.java`, and any other extra class you use. Also submit a `readme.txt` with any information the TA needs to run your code, in particular what are the parts that you have implemented and which parts are working (i,ii,iii,iv); explain any known bugs.

2. **Report:**
   A pdf file describing your implementation and your experiments.
   It should include:
   - Description of your chosen data structures, including the data structure used for your graph.
   - High-level description of the algorithms used to answer the questions in task item 2.
   - Examples of outputs obtained with your program answering the questions of part 2. Include at least 3 outputs for each part. Make outputs of part iii correspond to outputs of part ii, using N3 such that the shortest path changes time.

Marking criteria:
30% Report
Complete report containing discussion and result for the 3 questions i, ii, iii.
If not all parts are completed, the report weight will be proportional to the work completed.
70% Program and correctness
10% class design and documentation
20% code and correctness of question i
20% code and correctness of question ii
20% code and correctness of question iii
Optional bonus question iv : 20% bonus will only be considered if parts i, ii, iii are working.
For bonus question iv, we distribute 5% for the report part and 15% is for the code and correctness part.

Important remarks:
You are encouraged to write your own code from scratch. However, if you reuse some code found elsewhere for solving some subtasks (online, textbook or from our labs), you must give a reference to any external resource you used (e.g. provide the links where you download the code, or other references used). Even if you modify a code after, you have to specify that you used certain code or certain library as a starting point, even if you just consult it to guide you in your design. <u>Failure to provide adequate references will be considered academic fraud with serious consequences</u>.
In the same way, since this assignment is individual, you must make sure that nobody uses your code to write their solution. You can discuss general ideas with friends, but do not share or show specific parts of your code: <u>exchanging source codes also constitutes academic fraud.</u> Note that all submitted programs will be subject to a plagiarism detection software that measures similarities in the program structure.

Reference Assignment inspired by :
Michel Couprie, Gilles Bertrand, 'Graphes et algorithmes - TP 3',
https://perso.esiee.fr/~coupriem/Graphestp3/graphestp3.html, accessed Nov 14, 2017.

Optional Bonus Question iv (Challenge)

Please proceed at your own risk. This problem may be too complex for some students to solve in the given time.

Write a general algorithm, which should work for any input subway graph following the specifications of this assignment, not only for the given graph.

Your challenge is to answer the followin question:

iv) Determine the minimum number of subway lines that must be broken (not functioning) in order to disconnect at least two stations in differenting (functional) lines.
Print one station number that identifies each subway line that is not functioning, and give two station numbers in functioning lines for which there is no path between them.
Explain your method in the report.

Hint: You can build another graph where vertices correspond to subway lines and edges indicate that the subway lines are connected by some walking edge (there is an station in each line that are connected by an edge of weight -1). This graph can be used for answering this question by designing an appropriate algorithm to test many possible scenarios of non-functioning combinations of lines.