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).