Advanced C++ programming : RATP application

SOUALHI Takieddine

SORBONNE UNIVERSITY

Abstract

This document presents a brief review and explanation to the C++ semester project. in what follows, we will present and explain each implemented algorithm separately

$Keywords : C++, STL$

1 Introduction

Given a real RATP (Régie autonome des transports parisiens) database of the map of the parisian metro, we were asked to create a console application that determines the fastest path (in termes of time) between two given metro stations that are identified by their specific IDs. The goal is to use Djikstra algorithm and print the solution on the console as a set of intructions for the user.

2 Mile Stones of the project :

2.1 UML of the project

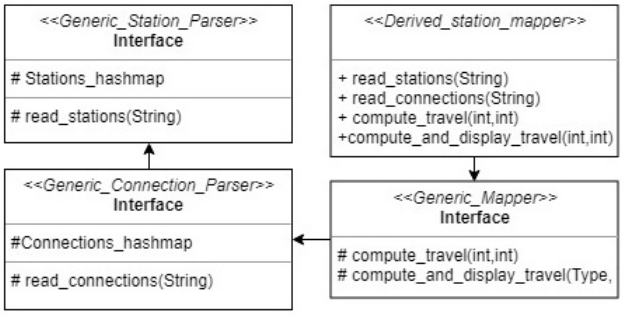

Figure 1: The Universe

2.2 Reading the "Stations.csv" file

This method which is a member of the class ¡¡Generic station parser¿¿ is implemented to read a csv file which contains all the nodes that will make up our graph. Several informations are associated with each node (a node represents one metro Station) : Station name, Station ID, Line name, Line ID and the Address of the line
To implement this method, we suggested algorithm demonstrated below

Result: Estimated Path
while *()* **do**
    1)Capture new frame Ik
    2) Extract and match features between Ik1 and Ik
    3) Compute essential matrix for image pair Ik1, Ik
    4) Decompose essentialmatrix into Rk and tk, andformTk
    5) Compute relative scale and rescale tk accordingly
    6) Concatenate transformation by computing Ck = Ck-1 x Tk
    7) Repeat from 1).
end
Close the file;

Algorithm 1: Read Stations algorithm

2.3 Generating the graph :

This method which is a member of the class ¡¡Generic connection parser¿¿ is implemented to read a csv file which contains the connections between the nodes and their costs. this methods writes the map of the parisian metro on the attribut of it's mother class
To implement this method, we suggested algorithm demonstrated below

Result: ConnectionsHashmap
input: FileName;
while *File not ended* **do**
    Read the Data into the file;
    Convert the starting station.ID to int;
    Convert the arrival station.ID to int;
    Convert the Cost of travel to int;
    Store the Data into StationsHashmap;
end
Close the file;

Algorithm 2: Read Connections algorithm

2.4 Dijkstra algorithm

Dijkstra's Algorithm works by visiting vertices in the graph starting with the object's starting point. It then repeatedly examines the closest not-yet-examined vertex, adding its vertices to the set of vertices to be examined. It expands outwards from the starting point until it reaches the goal. Dijkstra's Algorithm

is guaranteed to find a shortest path from the starting point to the goal, as long as none of the edges have a negative cost. (I write “a shortest path” because there are often multiple equivalently-short paths.).
The figure below shows a graphical demonstration of how the algorithm works [1]

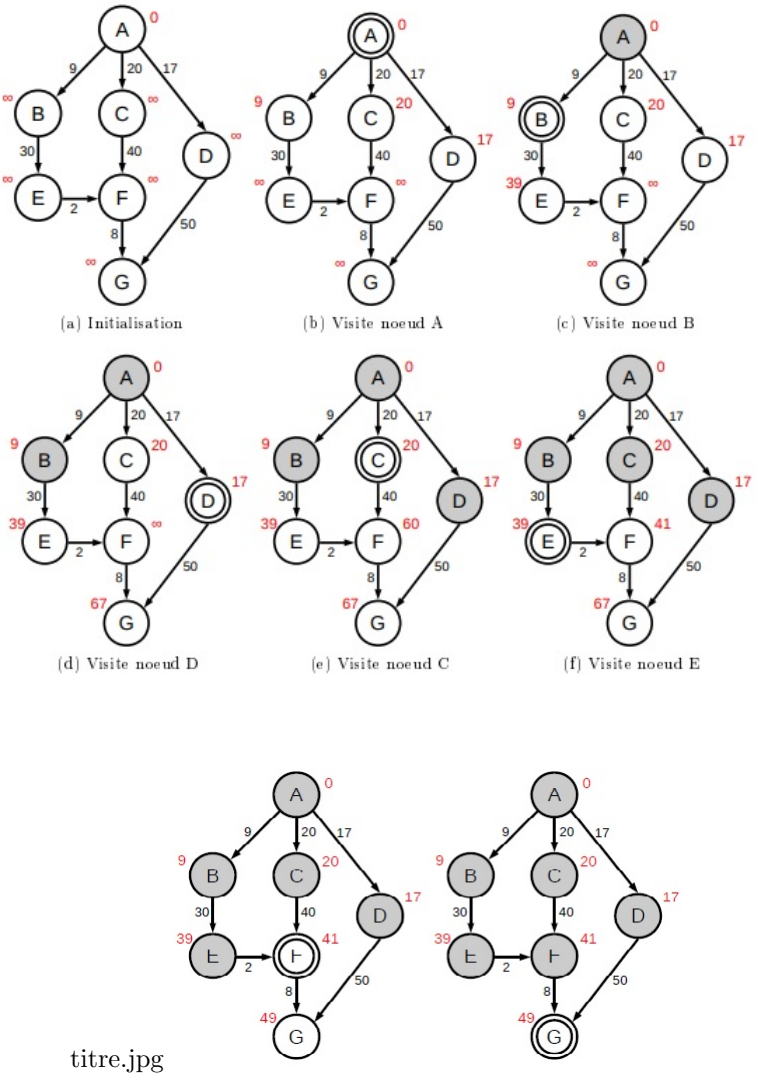

Figure 2: Graphic demonstration of Dijkstra algorithm

this algorithm was implemented in the method computeTravel() which is a member of the class ” Generic Mapper”. this method takes as input the IDs of the first and end nodes and generates in the output the shortest path between

the given inputs. the figure below shows the pseudo-code we used to implement the algorithm :

```
Result: ShortestPath,distanceMin[endNode
input: StartNode,endNode;
visitedNodes =0 ;
for Every Node n of the graphe do
    distanceMin[n]=inf;
end
while visitedNodes doesnt contain all nodes of G do
    CurrentNode=Not visited node with the smallest cost;
    VisitedNodes=VisitedNodes+CurrentNode;
    if CurrentNode==endNode then
        Break;
    end
    for Every Child Node n of the Current Node do
        distance=distanceMin[CurrentNode]+Distance[n];
        if distance¡distanceMin[n] then
            distanceMin[n]=distance;
            predecessor[n]=currentNode;
        end
    end
    ShortestPath=0
    if endNode is a predecessor then
        currentNode = endNode; Add (endNode to ShortestPath) while
         currentNode is a predecessor do
            Add (currentNode to the head of ShortestPath)
             currentNode=predecessor[currentNode
        end
        ;
    end
end
```

Algorithm 3: Djikstra Algorithm

2.5 Printing out the path "Compute-and-Display()"

The idea we used for this method is very simple, first of all we call the method compute-travel() to generate the shortest path between two given nodes, after that we proceed to determine the the IDs of the stations where the user has to change lines ,once this done, we proceed to printing the changes of the lines in the path, the corresponding stations and how much does it take to do each step

2.6 Upgrades

To realize the upgrades asked in the project subject, we suggested to use one of the metrics use for measuring the difference between two strings. We propose to

use the Levenshtein distance. in dynamic programming this method calculates the least number of operations to modify one string in order to get the other one.
1- A matrix is initialized measuring in the (m, n) cell the Levenshtein distance between the m-character prefix of one with the n-prefix of the other word.
2- The matrix can be filled from the upper left to the lower right corner.
3- Each jump horizontally or vertically corresponds to an insert or a delete, respectively.
4- The cost is normally set to 1 for each of the operations.
5-The diagonal jump can cost either one, if the two characters in the row and column do not match else 0, if they match. Each cell always minimizes the cost locally.
6-This way the number in the lower right corner is the Levenshtein distance between both words.

the figure below demonstrates an example that features the comparison of "HONDA" and "HYUNDAI":

		H	Y	U	N	D	A	I
	0	1	2	3	4	5	6	7
H	1	0	1	2	3	4	5	6
O	2	1	1	2	3	4	5	6
N	3	2	2	2	2	3	4	5
D	4	3	3	3	3	2	3	4
A	5	4	4	4	4	3	2	3

Figure 3: Graphic demonstration Livenstein distance

3 Conclusion

This project has been a great experience to put into practice the materials we acquired in lectures, and we were able to
-know how to choose the appropriate container to use
-how to exploit, create and manage .csv databases using C++
-Sticking to a software engineering specifications sheet
-Using the "Agile Technique " for project management
[1]

References

[1] Norvig P. Davis Russell, S. J. *Artificial intelligence: a modern approach. 3rd ed. Upper Saddle River*. NJ: Prentice Hall., 2010.