Tanay Cansin Dogan
150150119

BLG 336E: Assignment 2

Introduction:

In this assignment we have developed/implemented a path planning algorithm. In order to complete the task I had implemented the *Dijkstra's Algorithm* as return best paths from the source node to the each node according to the Dijkstra satisfaction. During the first submits, I couldn't complete the assignment since I had started to implement the assignment as Node-Edge model, when I encountered with the *"You must use the adjacency matrix representation for the graph construction"* expression it was a bit late so it was lack of "intersection avoidance" feature. Since HW2 is postponed I have revised the code and added the intersection avoidance. Now it is fully functional.

Compilation:

The code is written in c++11. Thus, should be compiled as: **g++ -std=c++11 150150119.cpp**

Functions and Methods:

- **Constructor:** Constructor takes "filename", then according to the node number N, constructs NxN matrix of zeros, and assigns Joseph's destination and hotel and Lucy's destination and hotel. Then parses each line of the file and commit them to the Matrix via **insertEdge()** method.
- **insertEdge():** takes a line as a string then parse it and commit it to the adjacency matrix. Each entry of the Adjacency matrix represents an edge with weight value, from row index to the column index.
- **ShowMatrix():** this method is a helper method to visualize the whole graph as a adjacency matrix and makes it easier to debug if an error happens.
- **in():** another helper function that indicates a node is in the vector or not.
- **calcPath():** this method calculates the path between given source and destination nodes, according to the *"Dijkstra Algorithm"*. It calculates paths for all of nodes as from the source node then return the only path of the destination node. In order to handle intersection problem it takes a *"bannedNode"* this is -1 by default which indicates any of the node is banned. But whether an intersection happened, this function called again with the intersection node as banned node and method ignore that node this time. If this function couldn't find a proper path to the destination it returns a path consists of only the source node so it can be understood that there is no path.
- **getTimeAcc():** this method takes a path then returns the arrival times corresponding to that path. It basically cumulatively sums weights of edges between previous and current nodes. And save the accumulated value.
- **intersectIn():** gets path vectors and corresponding time accumulation vectors for both Lucy and Joseph. Then check is there an intersection occurred. If intersection happened return the node that they intersect, otherwise returns -1 to indicate intersection didn't happen.
- **PrintResult():** Another helper method which prints the desired outputs in the assignment. This method makes this task more compact and improve the readability.

- **Homework():** This method is where the magic happened. It initially calculates paths from the Hotels and the Destinations, and from the Destinations to the Hotels for both Joseph and Lucy. Then check whether an intersection occurred or not between those paths (of course except the paths of same person). If intersection is happened (which can be understood by result is -1 or not) calculate the alternative paths again. Otherwise print them directly. If both of them has an alternative path chose the better one (compare the time it takes, chose lesser difference). If no alternative for the both of them. Print “No solution”.