# DSA Project Documentation

**By**:

NIHAL VIDYALA – 14BCE0453

MANORATH BAJAJ – 14BCE0442

PROF. PRABAKARAN N.

D2 SLOT

DATA STRUCTURE AND ALGORITHMS

# Abstract

Our project is a shortest distance calculator. The calculator is focused on locations within VIT including the subway tunnels, hostel blocks and academic blocks. Its purpose is to use the Floyd-Warshall algorithm to calculate the shortest distance between any two locations in VIT. In order to implement the algorithm, a graph has been made of VIT. The nodes of the graphs are buildings within VIT and the edges of the graph are the possible paths to travel between the buildings. The initial input is an adjacency and cost matrix that shows whether two node are adjacent as well as the weight of the edge connecting them. Using this matrix, the intermediate matrices are calculated and the final matrix which provides the shortest path between any two nodes in the graph is calculated. Rather than using estimated values of the distances between buildings in VIT, we will be using a Pedometer App to find the exact weights of the edges of our graph. This ensures that the result of our calculator is as accurate as possible. In addition, we will also be displaying the order in which the nodes need to be traversed in order to reach the destination in the least distance. We will have another function which will display the shortest path between two points assuming that you wish to pass through another point.

# Related Works

The best known example of a real life path finding program is Google Maps. Google Maps uses such algorithms to find the shortest path between any two given points across the world, while also containing updates on traffic conditions, blocked streets, etc. Google Maps uses the A\* algorithm to find the shortest path between two nodes. A\* is a generalized version of the Dijkstra’s algorithm.

# Project

## Proposal

The project will use the Floyd-Warshall algorithm to calculate the shortest path between two given nodes. The reason we are using the Floyd-Warshall algorithm over Dijkstra’s or the Bellman-Ford algorithm is because the latter two algorithms are single source shortest path (SSSP) algorithms. They can only calculate the minimum distance to other nodes from a given source node. While Dijkstra’s algorithm has running time O(V2) and Floyd-Warshall runs in O(V3) time (where V is the number of vertices), Floyd-Warshall calculates the shortest path matrix for all nodes in one run. Dijkstra’s algorithm requires V iterations to find the shortest path from any given vertex of the graph, which leads to O(V3) time.

## Model

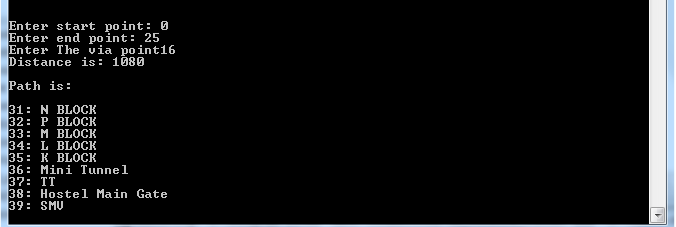
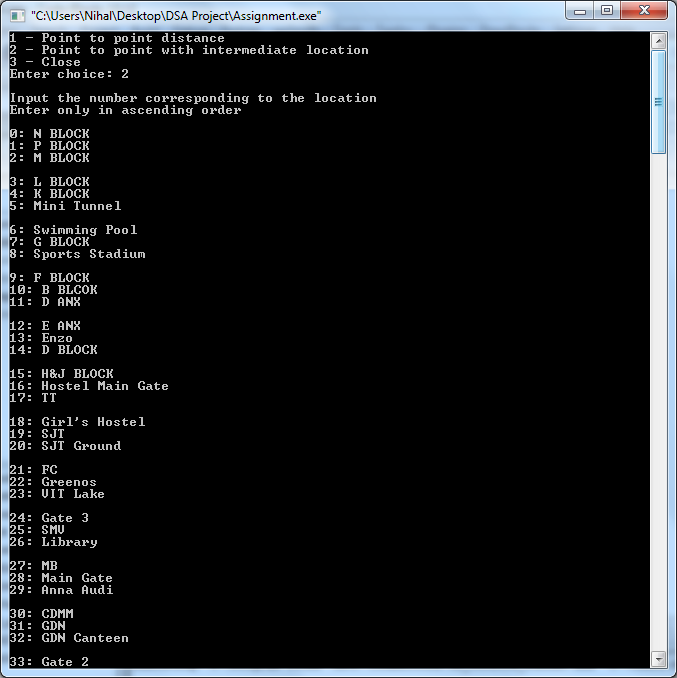
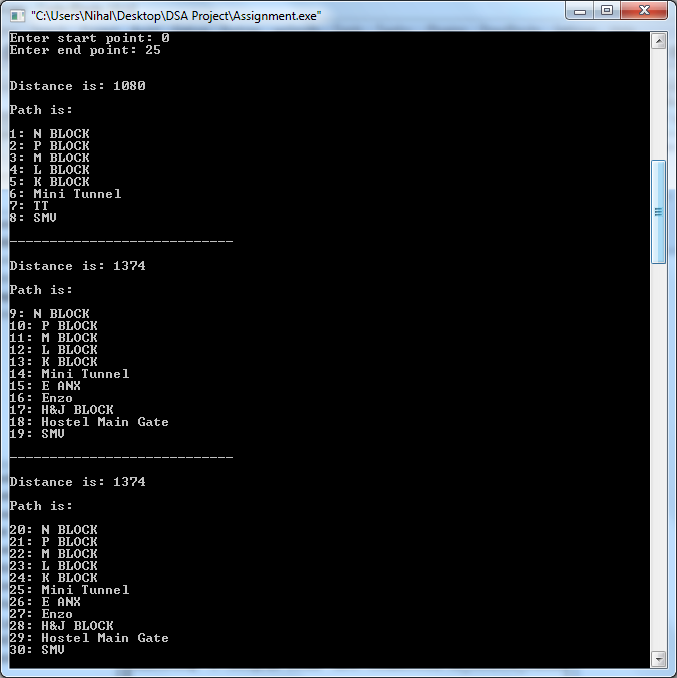
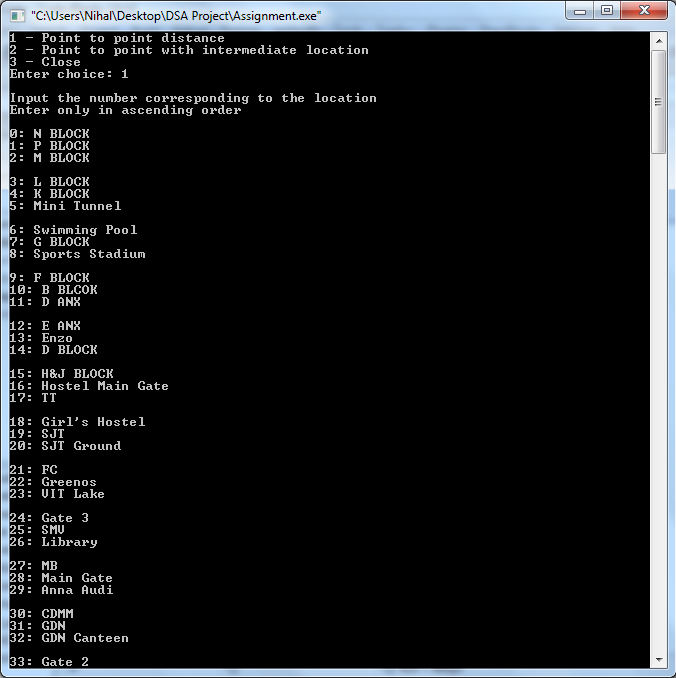
The program is structured as such: an input, a calculation and an output. The input from the user will require a start point and an end point. The calculation will involve finding the shortest path matrix. This matrix is calculated independent of the user’s input. The user’s input will be utilized in the calculation process to store in an array the order in which the vertices are traversed from the user’s start point to the destination. This is done to give the final path for the user. The final output provides both the path the user should take along with the distance calculated.

## Design

The program will have two files. One will be a container class that is used specifically for the calculation using the Floyd-Warshall algorithm. The other file will contain the input and output prompts for the user. This file will also contain another class which will store the vertices of the shortest path in an array.

# Implementation

Below are screenshots of the running of our program



# Challenges

While our program does properly calculate the distances between any two vertices in a graph, we had some difficulty creating our graph. As a result, some of the vertices of our graph are not entirely accurate representations of where the actual places are located within VIT. If we had added more nodes, it is likely that we could have overcome this problem.

Our program also does not display the correct path if the value of the starting vertex is greater than that of the end point vertex. It calculates the distance correctly and displays a general path (leaving out quite a few nodes). As such, the nodes must be entered in ascending order. If the nodes are entered in descending order, the program automatically exchanges the start and end points, giving you a reversed path. We have located the error within the printPath function, but we were not able to rectify the problem.

Another problem we experienced is that, very often, only two shortest paths will exist to a given point. This is naturally because of the layout of VIT. If a shorter path is found before a longer path, that longer path is completely disregarded and not stored by the intermediate matrices, as a better shortest path matrix exists already.

# Conclusion

After having tested several cases and even calculating their distances according to the graph we created, we have found that the graph does give accurate results for a majority of input values. The location of the nodes are not always at the actual locations that they indicate, which is why the output may appear strange for certain input values. As a whole, the program does work and provides an accurate result for the distance between buildings.