

Project Report



Group number 10

Group Members: Fatima Shibli, and Manoj Gurung

CSCE 3110-021

Instructor: Dr. Yuan Li

August 08, 2022

Index

Index ………………………………….…………………………..…… Page 1

Project goals …………………………………………….…………. Page 2

Explanation for the project ………………………..…..…... Page 2

Implemented Algorithms ……………………………...…….. Page 3

Code algorithm example …………………………….…..…... Page 4

Screen Shot from used code (1) ………………….………… Page 5

Screen Shot from used code (2) …………………….……… Page 6

The main aim or objective of this project was to answer the following questions:

1. I am in city "A", can I fly to city "B" with less than x connections? Give me the route with the smallest number of connections or tell me there is no such a route.

2. Give me the route with the smallest number of connections from city "A" to city "D" through city "B" and "C". (The order of "B" and "C" is not important). Or tell me there is no such a route

3. I want to start from city "A", visit all the cities that can be reached and then come back to "A" using as less connections as possible. Give me a route or tell me there is no such a route. (Note: once you come back to "A", you do not go out anymore).

4. I am in city "A", my friend John is in a different city "B", and my other friend Ann is in yet another different city "C". We want to find a city different from the three cities we are in to meet so that the total number of connections among three of us is minimized. Tell me the city we should fly to and the routes for us or tell me there is no such a city.

Finding the smallest number of connections between cities or nodes, finding if there was no such a route between cities, and finding the path with the least connections among three different cities proved to be more challenging than we had expected. To achieve our goals with this project, we successfully traversed the given nodes (cities) properly and made sure we did not return to some nodes after we had been there once. Besides that, we also had to choose a perfect data structure to store all the data we received from text files named "city. name" and "flight.txt" and process it accordingly to fulfill our objectives. Initially, we had chosen arrays to store all the data we extracted from the text files. But, as the array size is fixed, we faced a problem where every time we extract data from the given text files, sometimes those data were not being processed correctly because they were not adequately stored in arrays. So, we had to choose a vector as our vital data structure to hold the city names and flight routes from the text files. Moreover, vectors are very dynamic structures, allowing us some flexibility while working with varying cases of data inputs from the given text files.

Furthermore, we had to select different algorithms to achieve other objectives in the task. One of the algorithms we chose to implement was Depth First Search (DFS) over Breadth First Search (BFS) because when we did our research, we found DFS algorithms to be more effective and convenient for us to perform the tasks provided to us successfully as there are 140 different cities in the text file named city. Name, and more than 60 flight routes in the text file named flight.txt, we had no idea how far the target node would be from the source node. And for this purpose, our research indicated that DFS algorithms were more effective than BFS algorithms in worse cases where the target nodes seem to be really distant from the source node where we began our probing. In terms of speed of operation, too, we came to find out that DFS algorithms were less time-consuming than BFS algorithms. Since we had to perform probing on so many nodes from any given source node to a target node, we were glad to choose DFS algorithms as it would save our time and effort in achieving the results we were yearning for while working on this project. The time complexity for any DFS algorithm is given by big-oh O(V + E), where V refers to the graph's vertex, and E refers to the edges in the corresponding chart.

Another algorithm we implemented was the Hamiltonian path algorithm to achieve our project objectives. According to it, we used a backtracking algorithm to implement the given task at hand, i.e., to go from a source node (city) A to all of the other nodes (cities) and return to the source node while taking fewer connections as possible. This backtracking algorithm creates an empty path array, and then a vertex 0 is added. Then, starting with vertex 1, more vertices are added. Then, we check to see if a vertex is close to one that has already been added before adding it. If we discover such a vertex, we include it in the solution. We return false in the absence of a vertex. The time complexity for the Hamiltonian path algorithm was found to be big-oh O(n \* n!).

One of the other algorithms we implemented was Floyd Warshall Algorithm to solve the problems presented by this project successfully. At first, the Floyd Warshall algorithm was used to find a distance matrix (here, a 2d vector had been used to hold the matrix) which contains the distance (no. of connections) from each city to each connected city. Then we picked out the rows of those three cities we needed and found the column for which the sum was for these rows. This gave us the index of the city with the minor overall connection required and thus the best city to visit. Then we used the algorithm in task 1 (based on Depth First Search) to print the shortest path from each of these cities to the selected city. The time complexity for a Floyd Warshall algorithm is given by big-oh O(n3).

Below is a sneak peek of the codes belonging to the Floyd Warshall Algorithm:

void floydWarshall()

{

int i, j, k;

for (i = 0; i < graph.cities.size(); i++)

for (j = 0; j < graph.cities.size(); j++)

{

int a = graph.adjMat[i][j];

graph.distanceFloyd[i][j] = a;

}

for (k = 0; k < graph.cities.size(); k++)

{

// Pick all cities as source one by one

for (i = 0; i < graph.cities.size(); i++)

{

// Pick all cities as destination for the above picked source

for (j = 0; j < graph.cities.size(); j++)

{ // updating closure matrix

if (graph.distanceFloyd[i][j] > (graph.distanceFloyd[i][k] + graph.distanceFloyd[k][j]) && (graph.distanceFloyd[k][j] != INF && graph.distanceFloyd[i][k] != INF))

graph.distanceFloyd[i][j] = graph.distanceFloyd[i][k] + graph.distanceFloyd[k][j];

}

}

}

}

The for loop with two nested for loops inside has a time complexity of n3, and the outside for loop has a runtime complexity of n. Thus, the total time complexity being O(n3 + n) = O(n3).

Also Here is a sneak peek at the Hamiltonian Cycle:

Begin

if there is no edge between node(k-1) to v, then

return false

if v is already taken, then

return false

return true; //otherwise it is valid

End

Below are some screenshots depicting successful execution of the codes:

Text

Description automatically generated

Screen Shot (1)

Text

Description automatically generated

Screen Shot (2)