CIS7 Project Documentation Guide

In the documentation, provide at least 2 pages (single-space) that contains the following components of your course project:

* Team name, members.
* Project Information and details: (30 points)
* What problems are you solving in this project?
* What solutions are you implementing in the project?
* Provide explanation of calculations and algorithm implementation.
* What is the program objectives? Explain how your program is interacting with the user and its purpose.
* How is discrete structures implemented in the C++ program?
* What are the limitations of the program?
* Provide recommendation on improving the limitations of the program.
* Flowchart OR Pseudocode. (30 points)
* Write the pseudocode for the program, from start to finish. Be sure to include decision-making branching.
* If you choose to do flowchart, use standard shapes for flowchart, be sure to include decision-making branching. You can use web-based tool such as Draw.io to build your flowchart.

Anthony Mendoza

In this project i'll be solving what trips the specialist can take, the shortest path and cost, and creating a c++ program that represents the trip, low-cost, and shortest path. Some solution ill be implementing in the project is trying to find the shortest path and low cost trips and using the Floyd-Warshall algorithm. To start us off we first need to know what type of paths/routes the specialist can take. When we calcuate and start off we can see that the specialist can take different routes just to get to these 3 different places, But we also have to figure out which route will be best to take.

Riverside → Moreno Valley → Perris → Hemet

Riverside → Hemet → Moreno Valley → Perris

Riverside → Hemet → Perris → Moreno Valley

Riverside → Moreno Valley → Hemet → Perris

Riverside → Perris → Moreno Valley → Hemet

Riverside → Perris → Hemet → Moreno Valley



Based off the the graph we can tell the distants from each location. In order to make things easily we can organize this information.

Riverside to Moreno Valley: 16

Riverside to Perris: 24

Riverside to Hemet: 33

Moreno Valley to Perris: 18

Moreno Valley to Hemet: 26

Perris to Hemet: 30

Now we can figure out the shortest route to take by adding each provided distance to figure out the shortest route.

Riverside → Moreno Valley → Perris → Hemet

Distance: 16 + 18 + 30 = 64

Riverside → Moreno Valley → Hemet → Perris

Distance: 16 + 26 + 30 = 72

Riverside → Perris → Moreno Valley → Hemet

Distance: 24 + 18 + 26 = 68

Riverside → Perris → Hemet → Moreno Valley

Distance: 24 + 30 + 26 = 80

Riverside → Hemet → Moreno Valley → Perris

Distance: 33 + 26 + 18 = 77

Riverside → Hemet → Perris → Moreno Valley

Distance: 33 + 30 + 18 = 81

After we calucated the distance we can tell that the shortest distance from Riverside and the other location, it is best to go from Riverside to Moreno Valley to Perris and lastly Hemet since the total of distance is 64.

To create a C++ program to represnet the trips, low-cost and shortest path we would have to use an adjacency Matrix, Floyd-Warshall Algorithm, and a output. The Matrix represents the grpah that includes the cities and distances. The Floyd-Warshall would find the shortest paths between all pairs of cities. Lastly output would use both the other apporaches and print the matrix and the distance.

#include <iostream>

#include <vector>

#include <climits>

using namespace std;

const int INF = INT\_MAX;

void printMatrix(const vector<vector<int>>& matrix) {

for (const auto& row : matrix) {

for (int val : row) {

if (val == INF) cout << "INF ";

else cout << val << " ";

}

cout << endl;

}

}

int main() {

vector<vector<int>> adjMatrix = {

{0, 16, 24, 33},

{16, 0, 18, 26},

{24, 18, 0, 30},

{33, 26, 30, 0}

};

cout << "Adjacency Matrix:" << endl;

printMatrix(adjMatrix);

// Implementing Floyd-Warshall Algorithm for shortest paths

int n = adjMatrix.size();

vector<vector<int>> dist = adjMatrix;

for (int k = 0; k < n; ++k) {

for (int i = 0; i < n; ++i) {

for (int j = 0; j < n; ++j) {

if (dist[i][k] != INF && dist[k][j] != INF && dist[i][k] + dist[k][j] < dist[i][j]) {

dist[i][j] = dist[i][k] + dist[k][j];

}

}

}

}

cout << "Shortest Path Matrix:" << endl;

printMatrix(dist);

return 0;

}

This code allows us to see the shortest distants from the different routes.

To write the Pseudocode we have to define INF and initilalize the adjaceny matrix adjMatrix with the distance from the cities. After we then have to print the adjacency matrix using printMatrix function. For Floyd-Warshall algorithm we set n to the size of the adjancency matrix, use dist to initiailize the distance matrix with the values from the adjacney matrix. Latestly we use the printMatrix function again to print the shortest path matrix.

BEGIN

// Constants

DEFINE INF as a large number (infinity)

// Initialize adjacency matrix with distances between cities

CREATE adjMatrix as a 2D array with values:

[0, 16, 24, 33]

[16, 0, 18, 26]

[24, 18, 0, 30]

[33, 26, 30, 0]

// Print Adjacency Matrix

PRINT "Adjacency Matrix:"

CALL printMatrix(adjMatrix)

// Number of cities

SET n to size of adjMatrix

// Initialize distance matrix with adjacency matrix values

SET dist to adjMatrix

// Floyd-Warshall Algorithm for shortest paths

FOR k FROM 0 TO n-1 DO

FOR i FROM 0 TO n-1 DO

FOR j FROM 0 TO n-1 DO

IF dist[i][k] is not INF AND dist[k][j] is not INF AND dist[i][k] + dist[k][j] < dist[i][j] THEN

SET dist[i][j] to dist[i][k] + dist[k][j]

END IF

END FOR

END FOR

END FOR

// Print Shortest Path Matrix

PRINT "Shortest Path Matrix:"

CALL printMatrix(dist)

END