Team Name: CSS

Members:

1. Aurelia Juan Sindhunirmala
2. Aurelisa Juan Sindhunirmala
3. Brittany Hailee Lua Chan

Project Documentation

In this project, we are solving the Case 1 Problem about the Inland Empire Solar Sales Travel. The solution we are implementing in the project is to apply the methods we learned from unit 10 to find the shortest path and the most low-cost trips for the marketing specialist. We searched for the shortest path for the marketing specialist starting from Riverside and going to the three different cities (Moreno Valley, Perris, and Hemet) by adding the weight that was given in the travel route. We found out that the route Riverside-> Moreno Valley -> Perris -> Hemet is the shortest path and cheapest cost, with a total of 64. The solution that we implemented to be able to convert the process of finding the shortest and cheapest path, is the Dijkstra Algorithm. This algorithm allows us to find the shortest path between nodes, thus, enabling us to calculate the shortest path from the source node to the last possible node. In this case, the shortest path and cheapest cost are the same because in the real scenario, fewer miles traveled means less fuel consumed.

Elaborating on the algorithm implementation in this code, the main algorithm used is the Dijkstra algorithm to find the shortest path in a weighted path. First of all, the graph is represented using an adjacency list, with nodes and edges defined by structures. The 'Graph' class includes functions for creating a new adjacency list node, adding edges, and printing the adjacent list. The 'dijkstra' function, implements Dijkstra's algorithm in finding the shortest paths from a given source node to all other nodes. The program calculates and prints the cumulative shortest paths and their costs from each source node to the next node. The code outputs the adjacency list and, for each source node, displays the cost of the shortest path to the next node and the cumulative minimum cost.

This program's objective is to aid the user in finding the shortest path with the cheapest cost. This program will be useful to assist the user as nowadays, we need to be efficient with time and distance since fuel costs a lot more. Discrete structures are implemented in the C++ program by using the concepts of graphs, adjacency lists, matrices, and algorithms such as Dijkstra's Shortest Path Algorithm. The limitation of this program is that it is only programmed to start from Riverside, but this limitation can be addressed by adjusting the Dijkstra function that we created to allow the starting point, or the source node, to be changed by the city that the user inputs. Our approach is not to go back to the cities that have already been visited as it will not be as effective as if we only travel through the cities once. Additionally, we did not consider the shortest miles for the marketing specialist to go back home to Riverside.

Pseudocode:

```
//Libraries
include <iostream>
include <iomanip>
include <vector>

//Constants
const int INT_MAX = 1e6
```

```
//Data structure to store adjacency list nodes

struct Node

    int src, weight

    Node* dest


//Data structure to store graph edge

struct Edge

    int src, dest, weight


//Class for Graph

class Graph

    //Function to allocate new node of Adjacency List

    Node* getAdjListNode(int src, int weight, Node* dest)

        Node* newNode = new Node

        newNode->src = src

        newNode->weight = weight

        newNode->dest = dest

        return newNode



    //Function to add an edge to the graph

    void addEdge(vector<Node*>& adj, int src, int dest, int weight)
```

```
        adj[src] = getAdjListNode(dest, weight, adj[src])


    //Function to print the adjacency list
    void printList(vector<Node*>& adj, int V)
        print “Adjacency List (Source, Destination, Weight):”
        print “Riverside - 0, Moreno - 1, Perris - 2, Hemet - 3”
        for i from 0 to (V-1)
            print “Node ”, i,  “: ”
            Node* temp = adj[i]
            while temp != NULL
                print “ (”, i, “, ”, temp->src, “, ”, temp->weight, “) ” //outputs the
adjacencies of the node and the corresponding weight
                temp = temp->dest



//Function to find the vertex with the minimum distance value
int minDistance(vector<Node*>& adj, int dist[], bool visited[], int V)
    //Initialize min value
    int min = INT_MAX, min_index

    //Loop through all vertices
    for v from 0 to V-1
```

```
            //Check if the vertex v is not visited and has a smaller distance than the current
            minimum
            if not visited[v] and dist[v] < min
                //Update the minimum distance and the corresponding index
                min = dist[v]
                min_index = v

        //Return the index of the vertex with the minimum distance
        return min_index

    //Function to perform Dijkstra's algorithm
    void dijkstra(vector<Node*>& adj, int src, int dist[], bool visited[], int V)
        //Initialize distances and visited array
        for i from 0 to V-1
            dist[i] = INT_MAX
            visited[i] = false

        //Distance of source vertex to itself
        dist[src] = 0

        //Find shortest path for all vertices
        for count from 0 to V-2
            int u = minDistance(adj, dist, visited, V)
            visited[u] = true


            Node* temp = adj[u]

            while temp != NULL

                int v = temp->src

                if not visited[v] and dist[u] != INT_MAX and dist[u] + temp->weight <

                dist[v]

                    dist[v] = (dist[u] + temp->weight)

                temp = temp->dest


//Function to calculate the minimum cost and shortest path for all source nodes

void costShortest(vector<Node*>& adj, int V)

    int totCost = 0

    bool visited[V]


    for src from 0 to 2

        int dist[V]

        for i from 0 to V-1

            visited[i] = false


        dijkstra(adj, src, dist, visited, V)


        int dest = (src + 1) % V
        print “Node ”, src, “ -> ”, dest, “ = ”, dist[dest]


        totCost += dist[dest]

    print “Minimum cost =  ”, totCost

    print “Shortest path = ”, totCost


//Main function

int main()

    //Create a graph object

    Graph graph


    //Number of Nodes

    int V = 4


    //Array of pointers to Node to represent adjacency list

    vector<Node*> adj(V)


    //Edge array

    Edge edges[] = {

        {0, 1, 16}, {0, 2, 24}, {0, 3, 33},

        {1, 0, 16}, {1, 2, 18}, {1, 3, 26},

        {2, 0, 24}, {2, 1, 18}, {2, 3, 30},

        {3, 0, 33}, {3, 1, 26}, {3, 2, 30},
    }


    //Fill adjacency list

    for edge in edges

        graph.addEdge(adj, edge.src, edge.dest, edge.weight)


    //Print the adjacency list

    graph.printList(adj, V)


    //Finding shortest Path

    costShortest(adj, V)


    return 0
```