A1-B3-42

Amber Shukla

DAA LAB

PRACTICAL NO. 2

Aim: Construction of Minimum Spanning Tree.

PART-A

CODE:

```
#include <bits/stdc++.h>
using namespace std;
#define ll long long

int main(void)
{
    int size;

    cout << "Enter Number of Frekles on Back: ";
    cin >> size;

    vector<vector<int>> Graph(size, vector<int>(size, -1));

    for (int i = 0; i < size; i++)
    {
        for (int j = 0; j < size; j++)
        {
            if (i == j)
                continue;
            else if (Graph[j][i] != -1)
                Graph[i][j] = Graph[j][i];
            else
            {
                cout << "Enter Distance of Frekle " << i << " to Frekle " << j << ":
";
                cin >> Graph[i][j];
            }
        }
    }

    cout << "\n\nThe adjacency matrix of the graph is:\n"
```

```cpp
         << endl;
    ;

    for (int i = 0; i < size; i++)
    {
        for (int j = 0; j < size; j++)
        {
            if (Graph[i][j] == -1)
                Graph[i][j] = 0;

            cout << Graph[i][j] << "    ";
        }
        cout << endl;
    }
    cout << "\n\n";

    vector<int> near(size, 0);
    vector<pair<int, int>> MCST(size - 1);

    int min = INT_MAX;
    int u = 0, v = 0;
    for (int i = 0; i < size; i++)
    {
        for (int j = i; j < size; j++)
        {
            int weight = std::min(Graph[i][j], Graph[j][i]);
            if (weight && weight < min)
            {
                min = weight;
                u = i;
                v = j;
            }
        }
    }

    cout << "First Selected Edge: " << u << " - " << v << endl;

    MCST[0] = {v, u};

    for (int i = 0; i < size; i++)
    {
        if (i != u && i != v) {
            int weight_u = Graph[i][u] > 0 ? Graph[i][u] : INT_MAX;
            int weight_v = Graph[i][v] > 0 ? Graph[i][v] : INT_MAX;
            near[i] = (weight_u < weight_v) ? u : v;
        }
    }

    near[u] = near[v] = -1;
```

```
    cout << "Near Array after first edge selection: " << endl;
    for (int l = 0; l < near.size(); l++) {
        cout << near[l] << " ";
    }
    cout << "\n\n";

    for (int i = 1; i < size - 1; i++)
    {
        min = INT_MAX;
        int k;

        for (int j = 0; j < size; j++)
        {
            if (near[j] != -1)
            {
                int weight = std::min(Graph[j][near[j]], Graph[near[j]][j]);
                if (weight && weight < min)
                {
                    min = weight;
                    k = j;
                }
            }
        }

        cout << "\nSelected Edge " << i << ": " << near[k] << " - " << endl;

        MCST[i] = {near[k], k};
        near[k] = -1;

        for (int j = 0; j < size; j++)
        {
            if (near[j] != -1)
            {
                int curr_weight = Graph[j][near[j]] > 0 ? Graph[j][near[j]] :
INT_MAX;
                int new_weight = Graph[j][k] > 0 ? Graph[j][k] : INT_MAX;
                if (new_weight < curr_weight)
                {
                    near[j] = k;
                }
            }
        }

        cout << "Near Array after iteration " << i << ": " << endl;

        for (int l = 0; l < near.size(); l++)
        {
            cout << near[l] << " ";
```

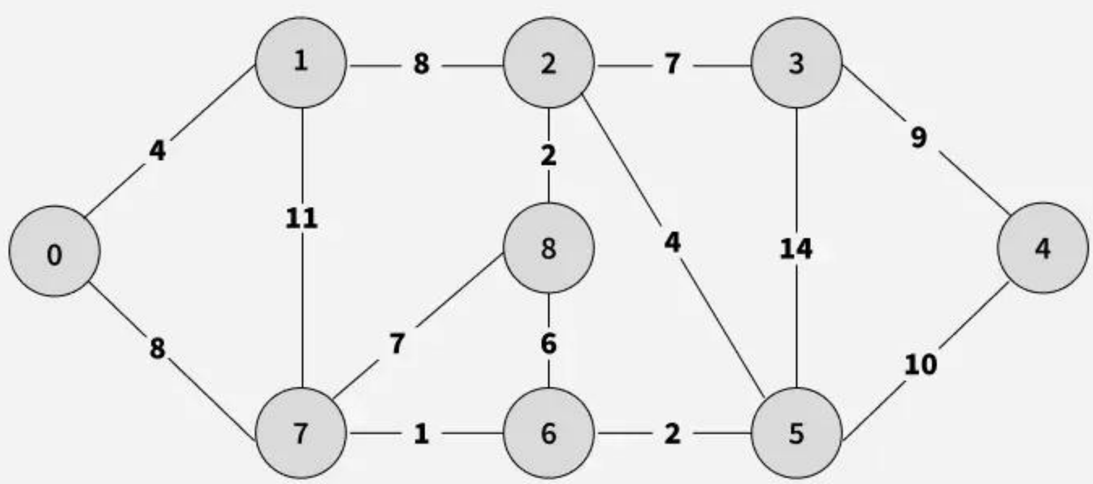
```
        }
        cout << "\n\n";
    }

    int totalWeight = 0;

    cout << "Edge \tWeight\n";
    for (int i = 0; i < size - 1; i++)
    {
        cout << MCST[i].first << " - " << MCST[i].second << "\t"
             << Graph[MCST[i].first][MCST[i].second] << "\n";
        totalWeight += Graph[MCST[i].first][MCST[i].second];
    }
    cout << "Total Weight of Minimum Spanning Tree: " << totalWeight << "\n";
}
```

OUTPUT:

```
Enter Number of Frekles on Back: 9
Enter Distance of Frekle 0 to Frekle 1: 4
Enter Distance of Frekle 0 to Frekle 2: 0
Enter Distance of Frekle 0 to Frekle 3: 0
Enter Distance of Frekle 0 to Frekle 4: 0
Enter Distance of Frekle 0 to Frekle 5: 0
Enter Distance of Frekle 0 to Frekle 6: 0
Enter Distance of Frekle 0 to Frekle 7: 8
Enter Distance of Frekle 0 to Frekle 8: 0
Enter Distance of Frekle 1 to Frekle 2: 8
Enter Distance of Frekle 1 to Frekle 3: 0
Enter Distance of Frekle 1 to Frekle 4: 0
Enter Distance of Frekle 1 to Frekle 5: 0
Enter Distance of Frekle 1 to Frekle 6: 0
Enter Distance of Frekle 1 to Frekle 7: 11
Enter Distance of Frekle 1 to Frekle 8: 0
Enter Distance of Frekle 2 to Frekle 3: 7
Enter Distance of Frekle 2 to Frekle 4: 0
Enter Distance of Frekle 2 to Frekle 5: 4
Enter Distance of Frekle 2 to Frekle 6: 0
Enter Distance of Frekle 2 to Frekle 7: 0
Enter Distance of Frekle 2 to Frekle 8: 2
Enter Distance of Frekle 3 to Frekle 4: 9
Enter Distance of Frekle 3 to Frekle 5: 14
Enter Distance of Frekle 3 to Frekle 6: 0
Enter Distance of Frekle 3 to Frekle 7: 0
Enter Distance of Frekle 3 to Frekle 8: 0
Enter Distance of Frekle 4 to Frekle 5: 10
Enter Distance of Frekle 4 to Frekle 6: 0
Enter Distance of Frekle 4 to Frekle 7: 0
Enter Distance of Frekle 4 to Frekle 8: 0
Enter Distance of Frekle 5 to Frekle 6: 2
Enter Distance of Frekle 5 to Frekle 7: 0
Enter Distance of Frekle 5 to Frekle 8: 0
Enter Distance of Frekle 6 to Frekle 7: 1
Enter Distance of Frekle 6 to Frekle 8: 6
Enter Distance of Frekle 7 to Frekle 8: 7
```

```
The adjacency matrix of the graph is:

0   4   0   0   0   0   0   8   0
4   0   8   0   0   0   0   11   0
0   8   0   7   0   4   0   0   2
0   0   7   0   9   14   0   0   0
0   0   0   9   0   10   0   0   0
0   0   4   14   10   0   2   0   0
0   0   0   0   0   2   0   1   6
8   11   0   0   0   0   1   0   7
0   0   2   0   0   0   6   7   0


First Selected Edge: 6 - 7
Near Array after first edge selection:
7 7 7 7 7 6 -1 -1 6


Selected Edge 1: 6 -
Near Array after iteration 1:
7 7 5 5 5 -1 -1 -1 6


Selected Edge 2: 5 -
Near Array after iteration 2:
7 2 -1 2 5 -1 -1 -1 2


Selected Edge 3: 2 -
Near Array after iteration 3:
7 2 -1 2 5 -1 -1 -1 -1


Selected Edge 4: 2 -
Near Array after iteration 4:
7 2 -1 -1 3 -1 -1 -1 -1
```

```
Selected Edge 5: 7 -
Near Array after iteration 5:
-1 0 -1 -1 3 -1 -1 -1 -1


Selected Edge 6: 0 -
Near Array after iteration 6:
-1 -1 -1 -1 3 -1 -1 -1 -1


Selected Edge 7: 3 -
Near Array after iteration 7:
-1 -1 -1 -1 -1 -1 -1 -1 -1

Edge     Weight
7 - 6    1
6 - 5    2
5 - 2    4
2 - 8    2
2 - 3    7
7 - 0    8
0 - 1    4
3 - 4    9
Total Weight of Minimum Spanning Tree: 37
PS C:\Users\amber\OneDrive\Desktop\RBU S3\DAA\P2>
```



Input Graph

PART-B:
Problem Statement:

A. Consider the following for deciding connections in same state in India:

i. Find the latitude and longitude of cities in same state. Consider 4 to 6 cities.

ii. Calculate the cost of connecting each pair of offices by computing the distance between different pair of different cities (as considered in part A) and construct a fully connected graph.

iii. Compute a minimum spanning tree using either Prims or Kruskals Method to find the cost of connecting offices in different cities.

CODE:

```
#include <algorithm>
#include <iostream>
#include <cmath>
#include <vector>
#include <string>
using namespace std;

const double unreachable = 999999.00;

double euclidean_dist(double x1, double y1, double x2, double y2);

int find(vector<int>& parent, int u) {
    if (parent[u] != u) {
        parent[u] = find(parent, parent[u]);
    }
    return parent[u];
}

void unionSets(vector<int>& parent, vector<int>& rank, int u, int v) {
    int rootU = find(parent, u);
    int rootV = find(parent, v);

    if (rootU != rootV) {
        if (rank[rootU] > rank[rootV]) {
            parent[rootV] = rootU;
        } else if (rank[rootU] < rank[rootV]) {
            parent[rootU] = rootV;
        } else {
            parent[rootV] = rootU;
            rank[rootU]++;
```

```
        }
    }
}

int main(void) {
    int cities;
    cout << "Enter number of cities (<= 6): ";
    cin >> cities;

    vector<string> city_names(cities);
    vector<vector<double>> geoloc(cities, vector<double>(2));

    for (int i = 0; i < cities; i++) {
        cout << "Enter Name Of City " << i + 1 << ": ";
        cin >> city_names[i];
        cout << "Enter Geolocation x of City " << i + 1 << ": ";
        cin >> geoloc[i][0];
        cout << "Enter Geolocation y of City " << i + 1 << ": ";
        cin >> geoloc[i][1];
    }

    vector<vector<double>> distances(cities, vector<double>(cities));

    for (int i = 0; i < cities; i++) {
        for (int j = 0; j < cities; j++) {
            if (i == j) {
                distances[i][j] = unreachable;
            } else {
                distances[i][j] = euclidean_dist(geoloc[i][0], geoloc[i][1],
geoloc[j][0], geoloc[j][1]);
            }
        }
    }

    vector<pair<double, pair<int, int>>> sorted_routes;

    for (int i = 0; i < cities; i++) {
        for (int j = i + 1; j < cities; j++) {
            if (distances[i][j] != unreachable) {
                sorted_routes.push_back({distances[i][j], {i, j}});
            }
        }
    }

    sort(sorted_routes.begin(), sorted_routes.end());

    vector<int> parent(cities);
    vector<int> rank(cities, 0);
    for (int i = 0; i < cities; i++) {
```

```
        parent[i] = i;
    }

    double total_cost = 0.0;
    vector<pair<string, string>> mst_edges;

    for (const auto& route : sorted_routes) {
        int u = route.second.first;
        int v = route.second.second;
        double dist = route.first;

        if (find(parent, u) != find(parent, v)) {
            unionSets(parent, rank, u, v);
            total_cost += dist;
            mst_edges.push_back({city_names[u], city_names[v]});
        }
    }

    cout << "\nMST Edges:\n";
    for (const auto& edge : mst_edges) {
        cout << edge.first << " - " << edge.second << endl;
    }
    cout << "Total cost of MST: " << total_cost << endl;

    return 0;
}

double euclidean_dist(double x1, double y1, double x2, double y2) {
    return sqrt(pow(y2 - y1, 2) + pow(x2 - x1, 2));
}
```

OUTPUT:

```
Enter number of cities (<= 6): 6
Enter Name Of City 1: Delhi
Enter Geolocation x of City 1: 28.704060
Enter Geolocation y of City 1: 77.102493
Enter Name Of City 2: Mumbai
Enter Geolocation x of City 2: 19.076090
Enter Geolocation y of City 2: 72.877426
Enter Name Of City 3: Kolkata
Enter Geolocation x of City 3: 22.572645
Enter Geolocation y of City 3: 88.363892
Enter Name Of City 4: Chennai
Enter Geolocation x of City 4: 13.067439
Enter Geolocation y of City 4: 80.237617
Enter Name Of City 5: Nagpur
Enter Geolocation x of City 5: 21.146633
Enter Geolocation y of City 5: 79.088860
Enter Name Of City 6: Bangalore
Enter Geolocation x of City 6: 12.971599
Enter Geolocation y of City 6: 77.594566

MST Edges:
Chennai - Bangalore
Mumbai - Nagpur
Mumbai - Bangalore
Delhi - Nagpur
Kolkata - Nagpur
Total cost of MST: 34.105
PS C:\Users\amber\OneDrive\Desktop\RBU S3\DAA\P2>
```