**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"

         << 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:

# A screenshot of a computer programA screenshot of a computer program AI-generated content may be incorrect.

**A diagram of a graph

AI-generated content may be incorrect.**

**PART-B:**

**Problem Statement:**

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

1. Find the latitude and longitude of cities in same state. Consider 4 to 6 cities.
2. 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.
3. 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:**

A screenshot of a computer

AI-generated content may be incorrect.