**Lab Assignment 7**

**Name = Ishwari Sahebrao Jeughale**

**Roll No =23527**

**Batch = S2**

#include <iostream>

using namespace std;

const int MAX\_VERTICES = 20;

const int MAX\_NAME\_LENGTH = 50;

const int INT\_MAX = 2147483647;

struct Edge

{

    int src, dest, weight;

};

struct AdjListNode

{

    int dest, weight;

    AdjListNode \*next;

};

class MyString

{

private:

    char str[MAX\_NAME\_LENGTH];

public:

    MyString()

    {

        str[0] = '\0';

    }

    void set(const char \*s)

    {

        int i = 0;

        while (s[i] != '\0' && i < MAX\_NAME\_LENGTH - 1)

        {

            str[i] = s[i];

            i++;

        }

        str[i] = '\0';

    }

    const char \*get() const

    {

        return str;

    }

    MyString substring(int start, int length) const

    {

        MyString result;

        int i;

        for (i = 0; i < length && str[start + i] != '\0'; i++)

        {

            result.str[i] = str[start + i];

        }

        result.str[i] = '\0';

        return result;

    }

};

class CampusGraph

{

private:

    int V;

    MyString departments[MAX\_VERTICES];

    int adjMatrix[MAX\_VERTICES][MAX\_VERTICES];

    AdjListNode \*adjList[MAX\_VERTICES];

    Edge edges[MAX\_VERTICES \* MAX\_VERTICES];

    int edgeCount;

    int find(int parent[], int i)

    {

        if (parent[i] != i)

        {

            parent[i] = find(parent, parent[i]);

        }

        return parent[i];

    }

    void unionSets(int parent[], int rank[], int x, int y)

    {

        int xroot = find(parent, x);

        int yroot = find(parent, y);

        if (rank[xroot] < rank[yroot])

        {

            parent[xroot] = yroot;

        }

        else if (rank[xroot] > rank[yroot])

        {

            parent[yroot] = xroot;

        }

        else

        {

            parent[yroot] = xroot;

            rank[xroot]++;

        }

    }

    int minKey(int key[], bool mstSet[])

    {

        int min = INT\_MAX, min\_index = 0;

        for (int v = 0; v < V; v++)

        {

            if (!mstSet[v] && key[v] < min)

            {

                min = key[v];

                min\_index = v;

            }

        }

        return min\_index;

    }

    void sortEdges()

    {

        for (int i = 0; i < edgeCount - 1; i++)

        {

            for (int j = 0; j < edgeCount - i - 1; j++)

            {

                if (edges[j].weight > edges[j + 1].weight)

                {

                    Edge temp = edges[j];

                    edges[j] = edges[j + 1];

                    edges[j + 1] = temp;

                }

            }

        }

    }

public:

    CampusGraph(int vertices) : V(vertices), edgeCount(0)

    {

        for (int i = 0; i < V; i++)

        {

            for (int j = 0; j < V; j++)

            {

                adjMatrix[i][j] = 0;

            }

        }

        for (int i = 0; i < V; i++)

        {

            adjList[i] = nullptr;

        }

    }

    void setDepartmentName(int index, const char \*name)

    {

        if (index >= 0 && index < V)

        {

            departments[index].set(name);

        }

    }

    void addEdge(int src, int dest, int weight)

    {

        adjMatrix[src][dest] = weight;

        adjMatrix[dest][src] = weight;

        AdjListNode \*newNode = new AdjListNode;

        newNode->dest = dest;

        newNode->weight = weight;

        newNode->next = adjList[src];

        adjList[src] = newNode;

        newNode = new AdjListNode;

        newNode->dest = src;

        newNode->weight = weight;

        newNode->next = adjList[dest];

        adjList[dest] = newNode;

        edges[edgeCount].src = src;

        edges[edgeCount].dest = dest;

        edges[edgeCount].weight = weight;

        edgeCount++;

    }

    void printAdjMatrix()

    {

        cout << "\nAdjacency Matrix Representation:\n";

        cout << "   ";

        for (int i = 0; i < V; i++)

        {

            cout << departments[i].substring(0, 3).get() << " ";

        }

        cout << "\n";

        for (int i = 0; i < V; i++)

        {

            cout << departments[i].substring(0, 3).get() << " ";

            for (int j = 0; j < V; j++)

            {

                cout << adjMatrix[i][j] << "   ";

            }

            cout << "\n";

        }

    }

    void printAdjList()

    {

        cout << "\nAdjacency List Representation:\n";

        for (int i = 0; i < V; i++)

        {

            cout << departments[i].get() << " -> ";

            AdjListNode \*temp = adjList[i];

            while (temp != nullptr)

            {

                cout << departments[temp->dest].get() << "(" << temp->weight << ") -> ";

                temp = temp->next;

            }

            cout << "NULL\n";

        }

    }

    void kruskalMST()

    {

        sortEdges();

        int parent[MAX\_VERTICES];

        int rank[MAX\_VERTICES];

        for (int i = 0; i < V; i++)

        {

            parent[i] = i;

            rank[i] = 0;

        }

        cout << "\nMinimum Spanning Tree using Kruskal's Algorithm:\n";

        int totalWeight = 0;

        for (int i = 0; i < edgeCount; i++)

        {

            int x = find(parent, edges[i].src);

            int y = find(parent, edges[i].dest);

            if (x != y)

            {

                cout << departments[edges[i].src].get() << " - "

                     << departments[edges[i].dest].get() << " : "

                     << edges[i].weight << "m\n";

                totalWeight += edges[i].weight;

                unionSets(parent, rank, x, y);

            }

        }

        cout << "Total distance of MST: " << totalWeight << "m\n";

    }

    void primMST()

    {

        int parent[MAX\_VERTICES];

        int key[MAX\_VERTICES];

        bool mstSet[MAX\_VERTICES];

        for (int i = 0; i < V; i++)

        {

            key[i] = INT\_MAX;

            mstSet[i] = false;

        }

        key[0] = 0;

        parent[0] = -1;

        for (int count = 0; count < V - 1; count++)

        {

            int u = minKey(key, mstSet);

            mstSet[u] = true;

            for (int v = 0; v < V; v++)

            {

                if (adjMatrix[u][v] && !mstSet[v] && adjMatrix[u][v] < key[v])

                {

                    parent[v] = u;

                    key[v] = adjMatrix[u][v];

                }

            }

        }

        cout << "\nMinimum Spanning Tree using Prim's Algorithm:\n";

        int totalWeight = 0;

        for (int i = 1; i < V; i++)

        {

            cout << departments[parent[i]].get() << " - " << departments[i].get()

                 << " : " << adjMatrix[i][parent[i]] << "m\n";

            totalWeight += adjMatrix[i][parent[i]];

        }

        cout << "Total distance of MST: " << totalWeight << "m\n";

    }

    ~CampusGraph()

    {

        for (int i = 0; i < V; i++)

        {

            AdjListNode \*current = adjList[i];

            while (current != nullptr)

            {

                AdjListNode \*temp = current;

                current = current->next;

                delete temp;

            }

        }

    }

};

int main()

{

    CampusGraph campus(6);

    campus.setDepartmentName(0, "CSE");

    campus.setDepartmentName(1, "ECE");

    campus.setDepartmentName(2, "Library");

    campus.setDepartmentName(3, "Canteen");

    campus.setDepartmentName(4, "Admin");

    campus.setDepartmentName(5, "Mech");

    campus.addEdge(0, 1, 100);

    campus.addEdge(0, 2, 150);

    campus.addEdge(1, 2, 80);

    campus.addEdge(1, 3, 120);

    campus.addEdge(2, 3, 70);

    campus.addEdge(2, 4, 180);

    campus.addEdge(3, 4, 90);

    campus.addEdge(3, 5, 200);

    campus.addEdge(4, 5, 140);

    int choice;

    do

    {

        cout << "\nCollege Campus Graph Operations:\n";

        cout << "1. Display Adjacency Matrix\n";

        cout << "2. Display Adjacency List\n";

        cout << "3. Find MST using Kruskal's Algorithm\n";

        cout << "4. Find MST using Prim's Algorithm\n";

        cout << "5. Exit\n";

        cout << "Enter your choice: ";

        cin >> choice;

        switch (choice)

        {

        case 1:

            campus.printAdjMatrix();

            break;

        case 2:

            campus.printAdjList();

            break;

        case 3:

            campus.kruskalMST();

            break;

        case 4:

            campus.primMST();

            break;

        case 5:

            cout << "Exiting program...\n";

            break;

        default:

            cout << "Invalid choice! Please try again.\n";

        }

    } while (choice != 5);

    return 0;

}

**GitHub link:-**

[https://github.com/VedantKaulgekar/CollegeDSAL/blob/main/Assignment\_ HYPERLINK "https://github.com/VedantKaulgekar/CollegeDSAL/blob/main/Assignment\_7.cpp"7 HYPERLINK "https://github.com/VedantKaulgekar/CollegeDSAL/blob/main/Assignment\_7.cpp".cpp](https://github.com/VedantKaulgekar/CollegeDSAL/blob/main/Assignment_7.cpp)