**Experiment No. 5**

Name : Shubham Kaspate

Batch : I3

Roll No : 20141241

// A C++ program for Prim's Minimum Spanning Tree (MST) algorithm. The program is for adjacency //matrix representation of the graph

#include <bits/stdc++.h>

using namespace std;

#define V 5

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

{

// Initialize min value

int min = INT\_MAX, min\_index;

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

if (mstSet[v] == false && key[v] < min)

min = key[v], min\_index = v;

return min\_index;

}

void printMST(int parent[], int graph[V][V])

{

cout << "Edge \tWeight\n";

for (int i = 1; i < V; i++)

cout << parent[i] << " - " << i << " \t"

<< graph[i][parent[i]] << " \n";

}

void primMST(int graph[V][V])

{

int parent[V];

int key[V];

bool mstSet[V];

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++) {

// Pick the minimum key vertex from the

// set of vertices not yet included in MST

int u = minKey(key, mstSet);

// Add the picked vertex to the MST Set

mstSet[u] = true;

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

// if graph[u][v] is smaller than key[v]

if (graph[u][v] && mstSet[v] == false

&& graph[u][v] < key[v])

parent[v] = u, key[v] = graph[u][v];

}

// print the constructed MST

printMST(parent, graph);

}

// Driver's code

int main()

{

int graph[V][V] = { { 0, 2, 0, 6, 0 },

{ 2, 0, 3, 8, 5 },

{ 0, 3, 0, 0, 7 },

{ 6, 8, 0, 0, 9 },

{ 0, 5, 7, 9, 0 } };

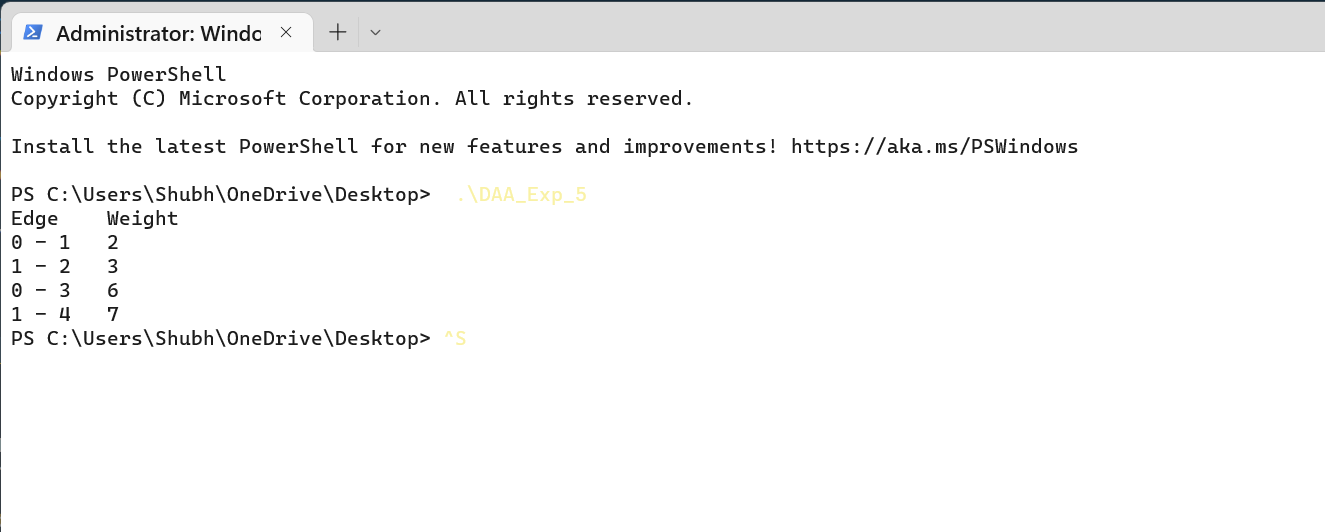
// Print the solution

primMST(graph);

return 0;

}

**Output :**

****

Kruskal’s algorithm :

#include <bits/stdc++.h>

using namespace std;

// DSU data structure

// path compression + rank by union

class DSU {

    int\* parent;

    int\* rank;

public:

    DSU(int n)

    {

        parent = new int[n];

        rank = new int[n];

        for (int i = 0; i < n; i++) {

            parent[i] = -1;

            rank[i] = 1;

        }

    }

    // Find function

    int find(int i)

    {

        if (parent[i] == -1)

            return i;

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

    }

    // Union function

    void unite(int x, int y)

    {

        int s1 = find(x);

        int s2 = find(y);

        if (s1 != s2) {

            if (rank[s1] < rank[s2]) {

                parent[s1] = s2;

                rank[s2] += rank[s1];

            }

            else {

                parent[s2] = s1;

                rank[s1] += rank[s2];

            }

        }

    }

};

class Graph {

    vector<vector<int> > edgelist;

    int V;

public:

    Graph(int V) { this->V = V; }

    void addEdge(int x, int y, int w)

    {

        edgelist.push\_back({ w, x, y });

    }

    void kruskals\_mst()

    {

        // 1. Sort all edges

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

        // Initialize the DSU

        DSU s(V);

        int ans = 0;

        cout << "Following are the edges in the "

                "constructed MST"

            << endl;

        for (auto edge : edgelist) {

            int w = edge[0];

            int x = edge[1];

            int y = edge[2];

            // Take this edge in MST if it does

            // not forms a cycle

            if (s.find(x) != s.find(y)) {

                s.unite(x, y);

                ans += w;

                cout << x << " -- " << y << " == " << w

                    << endl;

            }

        }

        cout << "Minimum Cost Spanning Tree: " << ans;

    }

};

// Driver's code

int main()

{

    /\* Let us create following weighted graph

                10

            0--------1

            | \  |

            6| 5\ |15

            |    \ |

            2--------3

                4    \*/

    Graph g(4);

    g.addEdge(0, 1, 10);

    g.addEdge(1, 3, 15);

    g.addEdge(2, 3, 4);

    g.addEdge(2, 0, 6);

    g.addEdge(0, 3, 5);

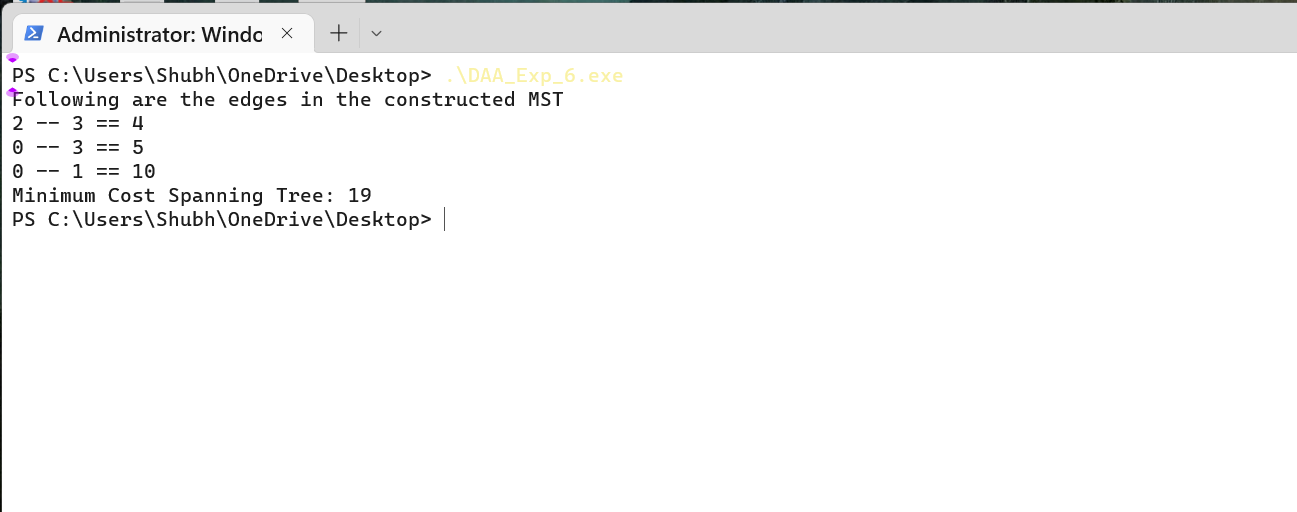
    // Function call

    g.kruskals\_mst();

    return 0;

}

**Output :**

****