Experiment-2.4

Aim of the Experiment :

Write a program to construct a MST (Minimum Spanning Tree) using:

a. Prim's Algorithm.

b. Kruskal's Algorithm

1. Problem Description :

Kruskal's Algorithm: In Kruskal's algorithm, sort all edges of the given graph in increasing order. Then it keeps on adding new edges and nodes in the MST if the newly added edge does not form a cycle. It picks the minimum weighted edge at first and the maximum weighted edge at last. Thus we can say that it makes a locally optimal choice in each step in order to find the optimal solution.

2. Algorithm :

KruskalMST(G)

```
for each vertex V in G do
        define a Cloud(v) of ←{v}
let Q be a priority queue.
Insert all edges into Q using their weights as the key
T ← Φ
while T has fewer than n-1 edges do
        edge e = T.removeMin()
        Let u, v be the endpoints of e
        if Cloud(v) ≠ Cloud(u) then
        Add edge e to T
        Merge Cloud(v) and Cloud(u)
return T
```

3. Complexity Analysis:

Time Complexity: O(E*logE) or O(E*logV)

Sorting of edges takes O(E*logE) time. The find and union operations can take at most O(logV) time. So overall complexity is O(E*logE + E*logV) time. The value of E can be at most O(V2), so O(logV) and O(logE) are the same. Therefore, the overall time complexity is O(E*logE) or O(E*logV).

Space Complexity: O(V + E), where V is the number of vertices and E is the number of edges in the graph.

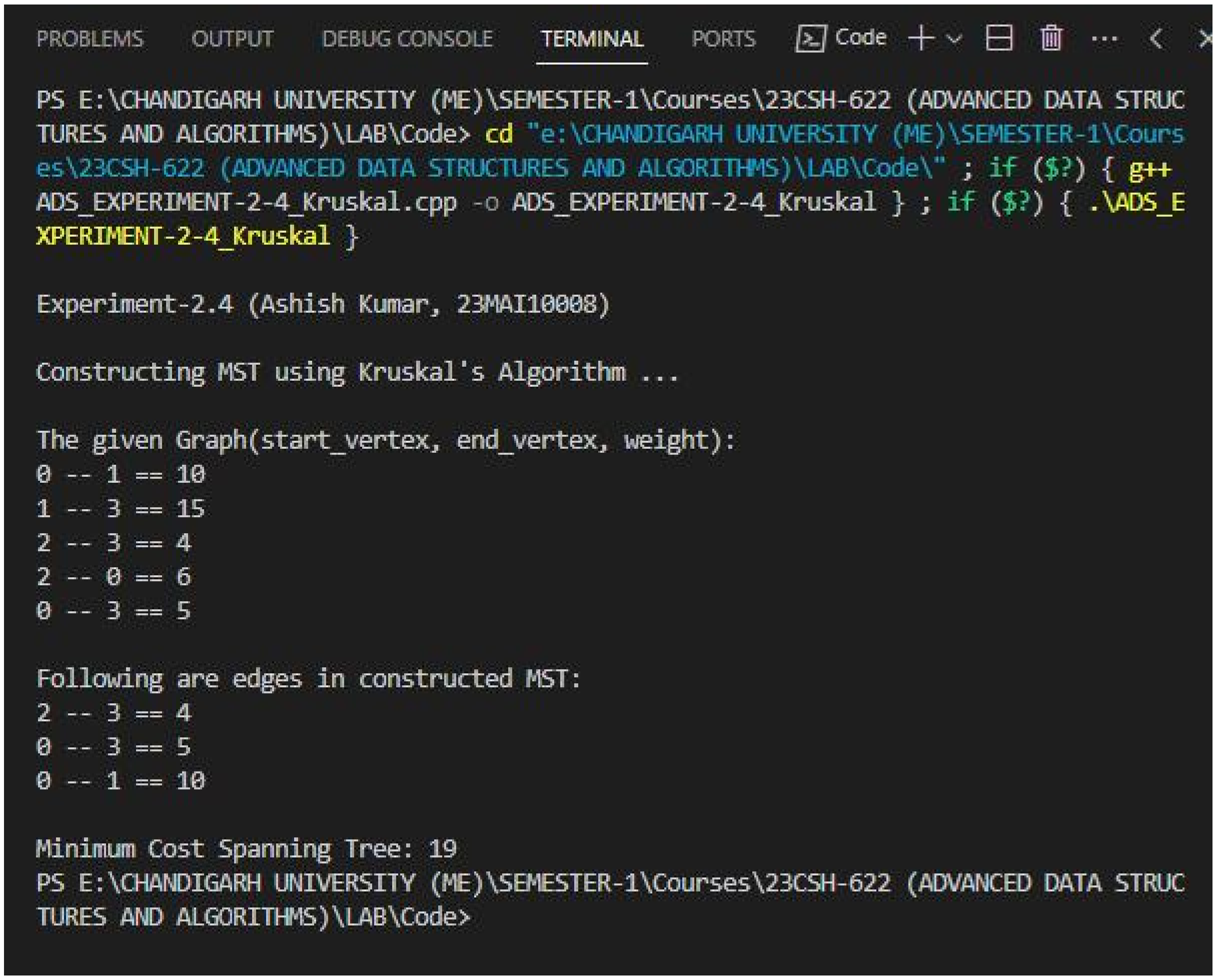
4. Pseudo Code :

```
MST- KRUSKAL (G, w)
Step 1. A ← Φ
Step 2. for each vertex v ∈ V [G]
Step 3. do MAKE - SET (v)
Step 4. sort the edges of E into non decreasing order by weight w
Step 5. for each edge (u, v) ∈ E, taken in non decreasing order by weight
Step 6. do if FIND-SET (μ) ≠ if FIND-SET (v)
Step 7. then A ← A ∪ {(u, v)}
Step 8. UNION (u, v)
Step 9. return A
```

5. Source Code for Experiment :

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

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;
            }
            else if (rank[s1] > rank[s2]) {
                parent[s2] = s1;
            }
            else {
                parent[s2] = s1;
                rank[s1] += 1;
            }
        }
    }

};

class Graph {
    vector<vector<int>> edgelist;
    int V;

public:
    Graph(int V) {
        this->V = V;
    }

    // Function to add edge in a graph
    void addEdge(int x, int y, int w) {
        edgelist.push_back({ w, x, y });
    }

    void kruskals_mst() {
        // Sort all edges
        sort(edgelist.begin(), edgelist.end());

        // Initialize the DSU
        DSU s(V);
        int ans = 0;
        cout << "\nFollowing are edges in constructed MST: "<< endl;
        for (auto edge : edgelist) {
            int w = edge[0];
            int x = edge[1];
            int y = edge[2];

            // Take 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 << "\nMinimum Cost Spanning Tree: " << ans <<endl;
    }

    // Print graph
    void printGraph() {
        cout << "\nThe given Graph(start_vertex, end_vertex, weight): "<<endl;
        for (auto edge : edgelist) {
            int w = edge[0];
            int x = edge[1];
            int y = edge[2];

            cout <<x<< " -- " <<y<< " == " <<w<< endl;
        }
    }
};

int main() {
    cout<<"\nExperiment-2.4 (Ashish Kumar, 23MAI10008)"<<endl<<endl;
    cout<<"Constructing MST using Kruskal's Algorithm ..."<<endl;
    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);
    // Print Graph
    g.printGraph();

    // Function call
    g.kruskals_mst();

    return 0;
}
```

6. Result/Output :



```
PS E:\CHANDIGARH UNIVERSITY (ME)\SEMESTER-1\Courses\23CSH-622 (ADVANCED DATA STRUC
TURES AND ALGORITHMS)\LAB\Code> cd "e:\CHANDIGARH UNIVERSITY (ME)\SEMESTER-1\Cours
es\23CSH-622 (ADVANCED DATA STRUCTURES AND ALGORITHMS)\LAB\Code\" ; if ($?) { g++
ADS_EXPERIMENT-2-4_Kruskal.cpp -o ADS_EXPERIMENT-2-4_Kruskal } ; if ($?) { .\ADS_E
XPERIMENT-2-4_Kruskal }

Experiment-2.4 (Ashish Kumar, 23MAI10008)

Constructing MST using Kruskal's Algorithm ...

The given Graph(start_vertex, end_vertex, weight):
0 -- 1 == 10
1 -- 3 == 15
2 -- 3 == 4
2 -- 0 == 6
0 -- 3 == 5

Following are edges in constructed MST:
2 -- 3 == 4
0 -- 3 == 5
0 -- 1 == 10

Minimum Cost Spanning Tree: 19
PS E:\CHANDIGARH UNIVERSITY (ME)\SEMESTER-1\Courses\23CSH-622 (ADVANCED DATA STRUC
TURES AND ALGORITHMS)\LAB\Code>
```

1. Problem Description :

Prim’s Algorithm: The Prim’s algorithm starts with an empty spanning tree. The idea is to maintain two sets of vertices. The first set contains the vertices already included in the MST, and the other set contains the vertices not yet included. At every step, it considers all the edges that connect the two sets and picks the minimum weight edge from these edges. After picking the edge, it moves the other endpoint of the edge to the set containing MST.

2. Algorithm :

PrimMST(G)

```
Consider a graph G=(V, E);
Let T be a tree consisting of only the starting vertex x;
while (T has fewer than I V I vertices)
{
    find a smallest edge connecting T to G-T;
    add it to T;
}
```

3. Complexity Analysis:

Time Complexity: O(V^2). If the input graph is represented using an adjacency list, then the time complexity of Prim’s algorithm can be reduced to O(E * logV) with the help of a binary heap.

Space Complexity: O(V).

4. Pseudo Code :

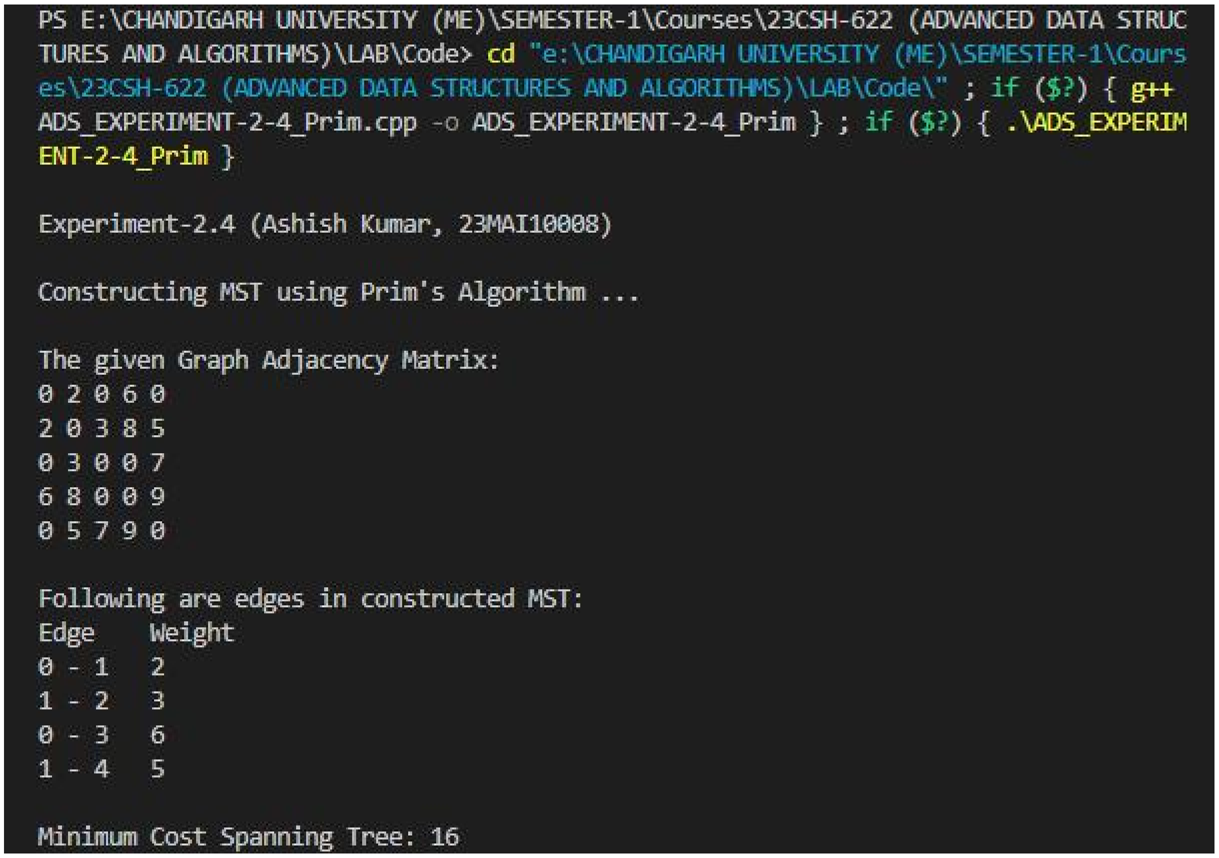
```
T = Φ;
U = { 1 };
while (U ≠ V)
    let (u, v) be the lowest cost edge such that u ∈ U and v ∈ V - U;
    T = T ∪ {(u, v)}
    U = U ∪ {v}
```

5. Source Code for Experiment :

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

// Number of vertices in the graph
#define V 5

void printGraph(int graph[V][V]) {
    cout<<"\nThe given Graph Adjacency Matrix: "<<endl;
    for (int i = 0; i < V; i++) {
        for (int j = 0; j < V; j++) {
            cout<<graph[i][j]<<" ";
        }
        cout<<endl;
    }
}

// Find the vertex with minimum key value
int minKey(int key[], bool mstSet[]) {

    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;
}


// Print the constructed MST stored in parent[]
void printMST(int parent[], int graph[V][V]) {

    int ans = 0;
    cout << "\nFollowing are edges in constructed MST: "<< endl;
    cout << "Edge \tWeight\n";
    for (int i = 1; i < V; i++) {
        cout << parent[i] << " - " << i << " \t"
             << graph[i][parent[i]] << " \n";
        ans += graph[i][parent[i]];
    }
    cout << "\nMinimum Cost Spanning Tree: " << ans <<endl;

}

// Construct and print MST for a graph
void primMST(int graph[V][V])
{
    // Array to store constructed MST
    int parent[V];

    // Key values used to pick minimum weight edge in cut
    int key[V];
    // Set of vertices included in MST
    bool mstSet[V];

    // Initialize all keys as INFINITE
    for (int i = 0; i < V; i++) {
        key[i] = INT_MAX, mstSet[i] = false;
    }

    // Always include first 1st vertex in MST.
    key[0] = 0;
    parent[0] = -1;

    // The MST will have V vertices
    for (int count = 0; count < V - 1; count++) {

        // Pick the minimum key vertex
        int u = minKey(key, mstSet);

        // Add the picked vertex to the MST Set
        mstSet[u] = true;

        // Update key value and parent index of adjacent vertices
        for (int v = 0; v < V; 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);
}
```

```
int main() {

    cout<<"\nExperiment-2.4 (Ashish Kumar, 23MAI10008)"<<endl<<endl;
    cout<<"Constructing MST using Prim's Algorithm ..."<<endl;

    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 Graph
    printGraph(graph);

    // Print the solution
    primMST(graph);

    return 0;
}
```

6. Result/Output :



```
PS E:\CHANDIGARH UNIVERSITY (ME)\SEMESTER-1\Courses\23CSH-622 (ADVANCED DATA STRUC
TURES AND ALGORITHMS)\LAB\Code> cd "e:\CHANDIGARH UNIVERSITY (ME)\SEMESTER-1\Cours
es\23CSH-622 (ADVANCED DATA STRUCTURES AND ALGORITHMS)\LAB\Code\" ; if ($?) { g++
ADS_EXPERIMENT-2-4_Prim.cpp -o ADS_EXPERIMENT-2-4_Prim } ; if ($?) { .\ADS_EXPERIM
ENT-2-4_Prim }

Experiment-2.4 (Ashish Kumar, 23MAI10008)

Constructing MST using Prim's Algorithm ...

The given Graph Adjacency Matrix:
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

Following are edges in constructed MST:
Edge    Weight
0 - 1   2
1 - 2   3
0 - 3   6
1 - 4   5

Minimum Cost Spanning Tree: 16
```

Learning outcomes (What I have learnt):

1. I learnt about how to input elements in an array.

2. I learnt about how to construct MST using Kruskal’s Algorithm.

3. I learnt about how to construct MST using Prim’s Algorithm.

4. I learnt about the differences between Kruskal’s and Prim’s Algorithm.

5. I learnt about time and space complexity of Kruskal’s and Prim’s Algorithm.