DAA Practical No: 09

Implement the following algorithm for minimum cost spanning tree

1)Prims algorithm using Binary Heap

2)Kruskal's algorithm using Min Heap

-- >

1) <u>Prims algorithm using Binary Heap</u>

Program Code:

```
// 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     // Number of vertices in the graph

// A utility function to find the vertex with minimum key value, from the set of vertices

// not yet included in MST

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

        int min = INT_MAX, min_index;       // Initialize min value

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

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

                        min = key[v], min_index = v;

        }

        return min_index;

}

// A utility function to print the constructed MST stored in parent[]

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";

        }

}
```

```
// Function to construct and print MST for a graph represented using adjacency matrix
representation

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

        int parent[V];     // Array to store constructed MST

        int key[V];    // Key values used to pick minimum weight edge in cut

        bool mstSet[V];    // To represent set of vertices included in MST

        // 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.

        // Make key 0 so that this vertex is picked as first vertex.

        key[0] = 0;

        parent[0] = -1;    // First node is always root of MST

        // The MST will have V vertices

        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;

        // Update key value and parent index of the adjacent vertices of the picked vertex.

        // Consider only those vertices which are not yet included in MST

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

        // graph[u][v] is non zero only for adjacent vertices of m mstSet[v] is false for vertices

        // not yet included in MST Update the key only 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:

2) <u>Kruskal's algorithm using Min Heap</u>

Program code:

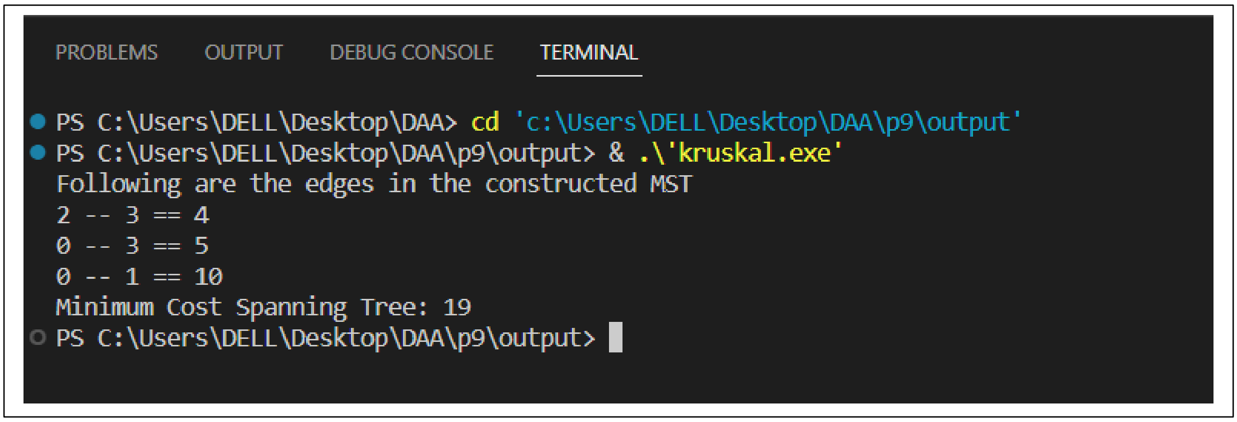
```
#include <bits/stdc++.h>
using namespace std;
// DSU data structure path compression + rank by union
class DSU {
	int* parent;
	int* rank;
public:
	DSU(int n){
		parent = new int[n];
		rank = new int[n];
		for (int i = 0; i < n; i++) {
```

```
                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;
                }
                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 << "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 code
int main(){
```

```
        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:



```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL

PS C:\Users\DELL\Desktop\DAA> cd 'c:\Users\DELL\Desktop\DAA\p9\output'
PS C:\Users\DELL\Desktop\DAA\p9\output> & .\'kruskal.exe'
Following are the edges in the constructed MST
2 -- 3 == 4
0 -- 3 == 5
0 -- 1 == 10
Minimum Cost Spanning Tree: 19
PS C:\Users\DELL\Desktop\DAA\p9\output>
```