**Experiment No. – 2.4**

**Aim**:

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

1. Prims Algorithm
2. Kruskal Algorithm
3. **Problem Description:**

We have given graph of n vertices we have to implement prims algorithm for constructing minimum spanning tree and calculate the shortest path in the MST and understand its time and space complexity. Prim’s algorithm is a [Greedy algorithm](https://www.geeksforgeeks.org/archives/18528). This algorithm always starts with a single node and moves through several adjacent nodes, in order to explore all of the connected edges along the way.

1. **Algorithm:**

**Initialization**

Pick a vertex ***r*** to be the root

Set ***D***(***r***) ***= 0, parent***(***r***) ***= null***

For all vertices ***v* ∈ *V***, ***v ≠ r***, set ***D***(***v***) ***= ∞***

Insert all vertices into priority queue ***P***,

Using distances as the keys

**While *P* is not empty:**

1. Select the next vertex ***u*** to add to the tree

***u = P.deleteMin***()

1. Update the weight of each vertex ***w*** adjacent to

***u*** which is **not** in the tree (i.e., ***w*** ∈ ***P***)

If ***weight***(***u, w***)**< *D***(***w***),

* 1. ***parent***(***w***) **= *u***
  2. ***D***(***w***) **= *weight***(***u, w***)
  3. Update the priority queue to reflect new distance for ***w***

1. **Complexity Analysis**

**Time Complexity: O(V2).** If the input [graph is represented using an adjacency list](https://www.geeksforgeeks.org/archives/27134), then the time complexity of Prim’s algorithm can be reduced to O(E \* logV) with the help of a binary heap.  In this implementation, we are always considering the spanning tree to start from the root of the graph  
**Auxiliary Space: O(V)**

1. **Pseudo Code**

**Pseudo Code for Prims Algorithm**

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

1. **Source Code (C/C++):**

**Code for Linear Search**

// The program is for adjacency matrix representation of the graph

#include <bits/stdc++.h>

using namespace std;

// Number of vertices in the graph

#define V 5

// 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[])

{

    // 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;

}

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

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

{

    int total\_weight = 0;

    cout << "Edge \tWeight\n";

    for (int i = 1; i < V; i++)

    {

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

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

        total\_weight += graph[i][parent[i]];

    }

    cout << "\nTotal cost of MST:" << total\_weight;

}

// Function to construct and print MST for

// a graph represented using adjacency

// matrix representation

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];

    // To represent 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.

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

    // vertex.

    key[0] = 0;

    // First node is always root of MST

    parent[0] = -1;

    // 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

    cout << "Name: Saurabh Kumar\n";

    cout << "UID : 23MAI10004\n\n";

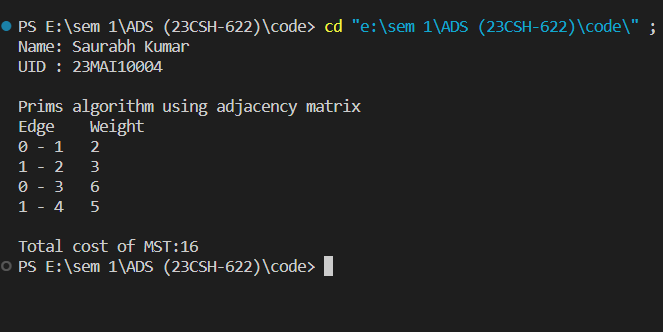
    cout << "Prims algorithm using adjacency matrix\n";

    primMST(graph);

    return 0;}

1. **Screenshot of Outputs:**

**Output for Prims Algorithm:**

****

1. **Problem Description:**

We have given graph of n vertices we have to implement **Kruskal's** algorithm for constructing minimum spanning tree and calculate the shortest path in the MST and understand its time and space complexity.The main target of the algorithm is to find the subset of edges by using which we can traverse every vertex of the graph. It follows the greedy approach that finds an optimum solution at every stage instead of focusing on a global optimum.

1. **Algorithm:**

Build a priority queue (min-based) with all of the edges of G.

T = Φ;

while(queue is not empty)

get minimum edge e from priorityQueue;

if(e does not create a cycle with edges in T)

add e to T;

return T;

1. **Complexity Analysis**

**Time Complexity: O(E logV),** here E is number of Edges and V is number of vertices in graph.  
**Auxiliary Space: O(V + E),** here V is the number of vertices and E is the number of edges in the graph.

1. **Pseudo Code**

**Pseudo Code for Kruskal's algorithm**

// Initialize result

mst\_weight = 0

// Create V single item sets

**for** **each** vertex v

parent[v] = v;

rank[v] = 0;

**Sort** all edges into non decreasing

order by weight w

**for** each (u, v) taken from the sorted list E

**do if** **FIND-SET**(u) != **FIND-SET**(v)

**print** edge(u, v)

mst\_weight += weight of edge(u, v)

**UNION**(u, v)

1. **Source Code (C/C++):**

**Code for Kruskal's algorithm**

// C++ program for Kruskal's algorithm

#include <bits/stdc++.h>

using namespace std;

// Creating shortcut for an integer pair

typedef pair<int, int> iPair;

// Structure to represent a graph

struct Graph{

    int V, E;

    vector<pair<int, iPair>> edges;

    // Constructor

    Graph(int V, int E)    {

        this->V = V;

        this->E = E;

    }

    // Utility function to add an edge

    void addEdge(int u, int v, int w){

        edges.push\_back({w, {u, v}});

    }

    // Function to find MST using Kruskal's

    // MST algorithm

    int kruskalMST();

};

// To represent Disjoint Sets

struct DisjointSets{

    int \*parent, \*rnk;

    int n;

    // Constructor.

    DisjointSets(int n)    {

        // Allocate memory

        this->n = n;

        parent = new int[n + 1];

        rnk = new int[n + 1];

        // Initially, all vertices are in

        // different sets and have rank 0.

        for (int i = 0; i <= n; i++)        {

            rnk[i] = 0;

            // every element is parent of itself

            parent[i] = i;

        }

    }

    // Find the parent of a node 'u'

    // Path Compression

    int find(int u)    {

        /\* Make the parent of the nodes in the path

        from u--> parent[u] point to parent[u] \*/

        if (u != parent[u])

            parent[u] = find(parent[u]);

        return parent[u];

    }

    // Union by rank

    void merge(int x, int y){

        x = find(x), y = find(y);

        /\* Make tree with smaller height

        a subtree of the other tree \*/

        if (rnk[x] > rnk[y])

            parent[y] = x;

        else // If rnk[x] <= rnk[y]

            parent[x] = y;

        if (rnk[x] == rnk[y])

            rnk[y]++;

    }

};

/\* Functions returns weight of the MST\*/

int Graph::kruskalMST(){

    int mst\_wt = 0;

    // Sort edges in increasing order on basis of cost

    sort(edges.begin(), edges.end());

    // Create disjoint sets

    DisjointSets ds(V);

    // Iterate through all sorted edges

    vector<pair<int, iPair>>::iterator it;

    for (it = edges.begin(); it != edges.end(); it++)    {

        int u = it->second.first;

        int v = it->second.second;

        int set\_u = ds.find(u);

        int set\_v = ds.find(v);

        // Check if the selected edge is creating

        // a cycle or not (Cycle is created if u

        // and v belong to same set)

        if (set\_u != set\_v){

            // Current edge will be in the MST

            // so print it

            cout << u << " - " << v << endl;

            // Update MST weight

            mst\_wt += it->first;

            // Merge two sets

            ds.merge(set\_u, set\_v);

        }

    }

    return mst\_wt;

}

int main(){

    int V = 9, E = 14;

    Graph g(V, E);

    g.addEdge(0, 1, 4);

    g.addEdge(0, 7, 8);

    g.addEdge(1, 2, 8);

    g.addEdge(1, 7, 11);

    g.addEdge(2, 3, 7);

    g.addEdge(2, 8, 2);

    g.addEdge(2, 5, 4);

    g.addEdge(3, 4, 9);

    g.addEdge(3, 5, 14);

    g.addEdge(4, 5, 10);

    g.addEdge(5, 6, 2);

    g.addEdge(6, 7, 1);

    g.addEdge(6, 8, 6);

    g.addEdge(7, 8, 7);

    cout << "Name: Saurabh Kumar\n";

    cout << "UID : 23MAI10004\n\n";

    cout << "Kruskal's algorithm \n";

    cout << "Edges of MST are \n";

    int mst\_wt = g.kruskalMST();

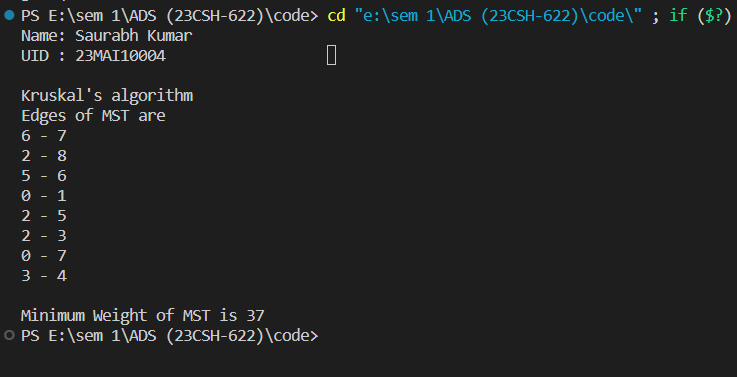
    cout << "\nMinimum Weight of MST is " << mst\_wt;

    return 0;

}

1. **Screenshot of Outputs:**

**Output for Kruskal's algorithm:**

****

1. **Learning Outcomes**
2. Learnt about Prim’s and Kruskal’s algorithm.
3. Learnt about time and space complexity of prim’s algorithm.
4. Learnt about time and space complexity of Kruskal’s algorithm.
5. Learnt about how to create and traverse a graph.
6. Learnt how to construct minimum spanning tree and calculate the minimum cost in the weighted graph using the Prim’s and Kruskal’s algorithms