**Pseudocode:**

This pseudocode assumes that the graph has a "vertex" class with a "parent" attribute, and an "edge" class with "u" and "v" attributes that represent the two endpoints of the edge, and a "weight" attribute that represents the weight of the edge. It also assumes that there are "make\_set", "find\_set", and "union" procedures available for manipulating the disjoint sets data structure.

The algorithm starts by adding all the edges of the graph to a priority queue, sorted by ascending weight. It then initializes the disjoint sets data structure with the vertices of the graph. It then repeatedly selects the edge with the minimum weight from the priority queue, and checks if the two endpoints of the edge belong to different sets. If they do, the edge is added to the result set and the two sets are united. The process continues until the result set contains |V|-1 edges, or the priority queue is empty. The result set is then returned as the minimum spanning tree of the graph.

PROCEDURE KruskalMST(graph: Graph)

result = new Set()

pq = new PriorityQueue()

FOR each edge in graph.edges

pq.add(edge)

FOR each vertex in graph.vertices

make\_set(vertex)

WHILE pq is not empty AND result.size() < graph.vertices.size() - 1

edge = pq.poll()

u = find\_set(edge.u)

v = find\_set(edge.v)

IF u != v

result.add(edge)

union(u, v)

return result

**Code:**

// Kruskal's algorithm to find Minimum

// Spanning Tree of a given connected, undirected and

// weighted graph

#include<bits/stdc++.h>

using namespace std;

typedef pair<int, int> iPair;

struct Graph

{

int V, E;

vector< pair<int, iPair> > edges;

Graph(int V, int E)

{

this->V = V;

this->E = E;

}

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

{

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

}

int kruskalMST();

};

struct DisjointSets

{

int \*parent, \*rnk;

int n;

DisjointSets(int n)

{

this->n = n;

parent = new int[n+1];

rnk = new int[n+1];

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

{

rnk[i] = 0;

parent[i] = i;

}

}

int find(int u)

{

if (u != parent[u])

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

return parent[u];

}

void merge(int x, int y)

{

x = find(x), y = find(y);

if (rnk[x] > rnk[y])

parent[y] = x;

else

parent[x] = y;

if (rnk[x] == rnk[y])

rnk[y]++;

}

};

int Graph::kruskalMST()

{

int mst\_wt = 0;

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

DisjointSets ds(V);

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);

if (set\_u != set\_v)

{

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

mst\_wt += it->first;

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, 1);

cout << "Edges of MST using Kruskal Algorithm are \n";

int mst\_wt = g.kruskalMST();

cout << "\nWeight of MST is " << mst\_wt;

return 0;

}

**Output Screenshot:**

