**Create a Graph Class**: Define a class to represent a graph with edges and vertices.

class Graph {

class Edge implements Comparable<Edge> {

int src, dest, weight;

public int compareTo(Edge compareEdge) {

return this.weight - compareEdge.weight;

}

}

int V, E; // V-> no. of vertices & E->no.of edges

Edge edge[]; // collection of all edges

// Creates a graph with V vertices and E edges

Graph(int v, int e) {

V = v;

E = e;

edge = new Edge[E];

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

edge[i] = new Edge();

}

}

**Sort the Edges**: Sort all edges in non-decreasing order of their weight.

Arrays.sort(graph.edge);

**Create Sets**: Create a set for each vertex to determine which vertices are included in the MST.

int[] parent = new int[graph.V];

for (int i = 0; i < graph.V; ++i)

parent[i] = i;

**Process Minimum Edges**: For each edge, check if it forms a cycle with the MST formed so far. If not, include it in the MST.

int e = 0; // Index used to pick the next edge

int i = 0; // Index used for sorted edges

for (i = 0; i < graph.E; i++) {

Edge next\_edge = graph.edge[i];

int x = find(parent, next\_edge.src);

int y = find(parent, next\_edge.dest);

if (x != y) {

result[e++] = next\_edge;

Union(parent, x, y);

}

}

**Define Find and Union Functions**: The find function determines the set to which a vertex belongs, and the union function joins two sets.

int find(int[] parent, int i) {

if (parent[i] == i)

return i;

return find(parent, parent[i]);

}

void Union(int[] parent, int x, int y) {

int xset = find(parent, x);

int yset = find(parent, y);

parent[xset] = yset;

}

.

**Output the MST**: The edges included in the MST are stored in the result array.

for (i = 0; i < e; ++i)

System.out.println(result[i].src + " -- " +

result[i].dest + " == " +

result[i].weight);