**ASSIGNMENT NO.3**

**Kruskal’s Algorithm:-**

import java.util.\*;

class Graph {

  class Edge implements Comparable<Edge> {

    int src, dest, weight;

    public int compareTo(Edge compareEdge) {

      return this.weight - compareEdge.weight;

    }

  };

  class subset {

    int parent, rank;

  };

  int vertices, edges;

  Edge edge[];

  Graph(int v, int e) {

    vertices = v;

    edges = e;

    edge = new Edge[edges];

    for (int i = 0; i < e; ++i)

      edge[i] = new Edge();

  }

  int find(subset subsets[], int i) {

    if (subsets[i].parent != i)

      subsets[i].parent = find(subsets, subsets[i].parent);

    return subsets[i].parent;

  }

  void Union(subset subsets[], int x, int y) {

    int xroot = find(subsets, x);

    int yroot = find(subsets, y);

    if (subsets[xroot].rank < subsets[yroot].rank)

      subsets[xroot].parent = yroot;

    else if (subsets[xroot].rank > subsets[yroot].rank)

      subsets[yroot].parent = xroot;

    else {

      subsets[yroot].parent = xroot;

      subsets[xroot].rank++;

    }

  }

  void KruskalAlgo() {

    Edge result[] = new Edge[vertices];

    int e = 0;

    int i = 0;

    for (i = 0; i < vertices; ++i)

      result[i] = new Edge();

    Arrays.sort(edge);

    subset subsets[] = new subset[vertices];

    for (i = 0; i < vertices; ++i)

      subsets[i] = new subset();

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

      subsets[v].parent = v;

      subsets[v].rank = 0;

    }

    i = 0;

    while (e < vertices - 1) {

      Edge next\_edge = new Edge();

      next\_edge = edge[i++];

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

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

      if (x != y) {

        result[e++] = next\_edge;

        Union(subsets, x, y);

      }

    }

    for (i = 0; i < e; ++i)

      System.out.println(result[i].src + " - " + result[i].dest + ": " + result[i].weight);

  }

  public static void main(String[] args) {

    int vertices = 6;

    int edges = 8;

    Graph G = new Graph(vertices, edges);

    G.edge[0].src = 0;

    G.edge[0].dest = 1;

    G.edge[0].weight = 4;

    G.edge[1].src = 0;

    G.edge[1].dest = 2;

    G.edge[1].weight = 4;

    G.edge[2].src = 1;

    G.edge[2].dest = 2;

    G.edge[2].weight = 2;

    G.edge[3].src = 2;

    G.edge[3].dest = 3;

    G.edge[3].weight = 3;

    G.edge[4].src = 2;

    G.edge[4].dest = 5;

    G.edge[4].weight = 2;

    G.edge[5].src = 2;

    G.edge[5].dest = 4;

    G.edge[5].weight = 4;

    G.edge[6].src = 3;

    G.edge[6].dest = 4;

    G.edge[6].weight = 3;

    G.edge[7].src = 5;

    G.edge[7].dest = 4;

    G.edge[7].weight = 3;

    G.KruskalAlgo();

  }

}

**output:-**

2 - 5: 2

2 - 3: 3

3 - 4: 3

0 - 1: 4