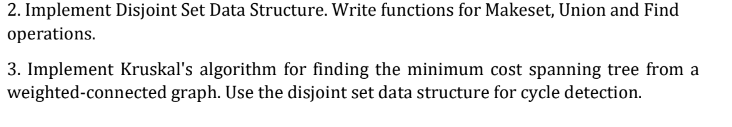
GRADED ASSIGNMENT-2

SUBMITTED BY:D Veera Harsha Vardhan Reddy/210C2030061



CODE:

#include <iostream>

using namespace std;

class DisjSet {

    int \*rank, \*parent, n;

public:

    DisjSet(int n)

    {

        rank = new int[n];

        parent = new int[n];

        this->n = n;

        makeSet();

    }

    void makeSet()

    {

        for (int i = 0; i < n; i++) {

            parent[i] = i;

        }

    }

    int find(int x)

    {

        if (parent[x] != x) {

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

        }

        return parent[x];

    }

    void Union(int x, int y)

    {

        int xset = find(x);

        int yset = find(y);

        if (xset == yset)

            return;

        if (rank[xset] < rank[yset]) {

            parent[xset] = yset;

        }

        else if (rank[xset] > rank[yset]) {

            parent[yset] = xset;

        }

        else {

            parent[yset] = xset;

            rank[xset] = rank[xset] + 1;

        }

    }

};

int main()

{

    DisjSet obj(6);

    obj.Union(0,2);

    obj.Union(0,4);

    obj.Union(1,2);

    obj.Union(4,5);

    if(obj.find(2)== obj.find(0)){

        cout<<"yes"<<endl;

    }

    else

    {

        cout<<"no"<<endl;

    }

    if(obj.find(4)== obj.find(0)){

        cout<<"yes"<<endl;

    }

    else

    {

        cout<<"no"<<endl;

    }

    if(obj.find(2)== obj.find(1)){

        cout<<"yes"<<endl;

    }

    else

    {

        cout<<"no"<<endl;

    }

    if(obj.find(3)==obj.find(5)){

        cout<<"yes"<<endl;

    }

    else

    {

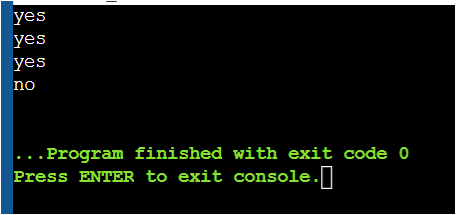
        cout<<"no"<<endl;

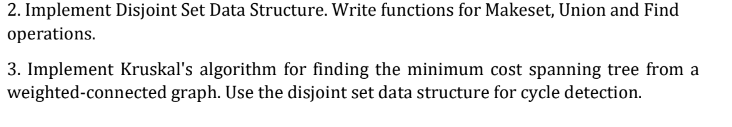
    }

    return 0;

}

OUTPUT:





CODE:

#include <bits/stdc++.h>

using namespace std;

class DisjSet {

    int \*rank, \*parent, n;

public:

    DisjSet(int n)

    {

        rank = new int[n];

        parent = new int[n];

        this->n = n;

        makeSet();

    }

    void makeSet()

    {

        for (int i = 0; i < n; i++) {

            parent[i] = i;

        }

    }

    int find(int x)

    {

        if (parent[x] != x) {

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

        }

        return parent[x];

    }

    void Union(int x, int y)

    {

        int xset = find(x);

        int yset = find(y);

        if (xset == yset)

            return;

        if (rank[xset] < rank[yset]) {

            parent[xset] = yset;

        }

        else if (rank[xset] > rank[yset]) {

            parent[yset] = xset;

        }

        else {

            parent[yset] = xset;

            rank[xset] = rank[xset] + 1;

        }

    }

};

class Graph {

    vector<vector<int> > edgelist;

    int V;

public:

    Graph(int V) { this->V = V; }

    void addEdge(int x, int y, int w)

    {

        edgelist.push\_back({ w, x, y });

    }

    void kruskals\_mst()

    {

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

        DisjSet 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];

            if (s.find(x) != s.find(y)) {

                s.Union(x, y);

                ans += w;

                cout << x << " -- " << y << " == " << w

                     << endl;

            }

        }

        cout << "Minimum Cost Spanning Tree: " << ans;

    }

};

int main()

{

    Graph a(9);

    a.addEdge(0, 1, 8);

    a.addEdge(1, 2, 4);

    a.addEdge(0, 3, 1);

    a.addEdge(1, 4, 2);

    a.addEdge(2, 5, 4);

    a.addEdge(4, 5, 9);

    a.addEdge(5, 1, 4);

    a.addEdge(2, 1, 9);

    a.addEdge(4, 0, 1);

    a.kruskals\_mst();

    return 0;

}

OUTPUT:

