**Name**: Ravi Jamanbhai Makwana

**Roll No**: 21BCP418

Div-6, G-12

**Assignment-6**

**Aim:**

Write a program to perform Kruskals using Union Find.

**Theory:**

• In Kruskal’s algorithm, sort all edges of the given graph in increasing order. Then it keeps on adding new edges and nodes in the MST if the newly added edge does not form a cycle. It picks the minimum weighted edge at first at the maximum weighted edge at last. Thus we can say that it makes a locally optimal choice in each step in order to find the optimal solution. Hence this is a Greedy Algorithm.

**Algorithm:**

1. Sort all the edges in non-decreasing order of their weight.

2. Pick the smallest edge. Check if it forms a cycle with the spanning tree formed so far. If the cycle is not formed, include this edge. Else, discard it.

3. Repeat step#2 until there are (V-1) edges in the spanning tree.

**Program:**

#include <stdio.h>

typedef struct Edges{

    int start,end,weight;

}Edges;

typedef struct Graph{

    //number of vertices

    int v;

    //number of edges

    int e;

    //set of edges

    Edges Edges[];

}Graph;

typedef struct subset{

    int parent;

}subset;

typedef struct size\_of\_each\_subset{

    int size;

}size\_of\_each\_subset;

//function to find parent of a node

int find(struct subset subset[], int x){

    if ((subset[x].parent) != x){

        return (find(subset, subset[x].parent));

    }

    return x;

}

void unionSets(int u, int v, struct subset s[], struct size\_of\_each\_subset r[]){

    if ((r[u].size) > (r[v].size)){

        s[v].parent = u;

        r[u].size += r[v].size;

    }

    else if ((r[u].size) < (r[v].size)){

        s[u].parent = v;

        r[v].size += r[u].size;

    }

    else{

        s[v].parent = u;

        r[u].size += r[v].size;

    }

}

void sort(struct Graph \*g, int no\_of\_edges){

    int startArr[(no\_of\_edges)];

    int endArr[(no\_of\_edges)];

    int weightArr[(no\_of\_edges)];

    for (int i=1;i<no\_of\_edges;i++){

        startArr[i] = g->Edges[i].start;

        endArr[i] = g->Edges[i].end;

        weightArr[i] = g->Edges[i].weight;

    }

    for (int i=1;i<no\_of\_edges;i++){

        int smallest = weightArr[i];

        int index = i;

        for (int j=i;j<no\_of\_edges;j++){

            if (weightArr[j] < smallest){

                smallest = weightArr[j];

                index = j;

            }

        }

        //moddify the arrays (swap values)

        int temp = startArr[i];

        startArr[i] = startArr[index];

        startArr[index] = temp;

        temp = endArr[i];

        endArr[i] = endArr[index];

        endArr[index] = temp;

        temp = weightArr[i];

        weightArr[i] = weightArr[index];

        weightArr[index] = temp;

    }

    //modify the graph

    int counter = 1;

    for (int i=1;i<no\_of\_edges;i++){

        g->Edges[counter].start = startArr[i];

        g->Edges[counter].end = endArr[i];

        g->Edges[counter].weight = weightArr[i];

        counter++;

    }

}

int kruskalAlgo(struct Graph \*g, int no\_of\_edges, struct subset s[], struct size\_of\_each\_subset r[]){

    int minCost = 0;

    ///sort the graph

    sort(g, no\_of\_edges);

    printf("\nThe edges in MST are:\n");

    for (int i=1;i<=(g->e);i++){

        int v1 = g->Edges[i].start;

        int v1\_parent = find(s, v1);

        int v2 = g->Edges[i].end;

        int v2\_parent = find(s, v2);

        int wt = g->Edges[i].weight;

        // If the parents are different that

        // means they are in different sets so

        // union them

        if (v1\_parent != v2\_parent){

            //union them

            unionSets(v1\_parent, v2\_parent, s, r);

            minCost+=wt;

            printf("%d--%d: %d\n",g->Edges[i].start, g->Edges[i].end, g->Edges[i].weight);

        }

    }

    return minCost;

}

int main(){

    Graph g;

    g.e=5;

    g.v=4;

    int no\_of\_vertices = 5; //same as (g.v)+1

    int no\_of\_edges = 6; //same as (g.e)+1

    subset s[5];

    size\_of\_each\_subset r[5];

    for (int a=1;a<no\_of\_vertices;a++){

        s[a].parent = a;

    }

    for (int b=1;b<no\_of\_vertices;b++){

        r[b].size = 1;

    }

    g.Edges[1].start = 1;

    g.Edges[1].end = 2;

    g.Edges[1].weight = 10;

    g.Edges[2].start = 2;

    g.Edges[2].end = 4;

    g.Edges[2].weight = 15;

    g.Edges[3].start = 3;

    g.Edges[3].end = 4;

    g.Edges[3].weight = 4;

    g.Edges[4].start = 1;

    g.Edges[4].end = 3;

    g.Edges[4].weight = 6;

    g.Edges[5].start = 1;

    g.Edges[5].end = 4;

    g.Edges[5].weight = 5;

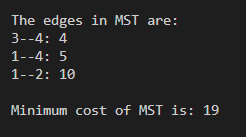
    int minCost = kruskalAlgo(&g, no\_of\_edges, s, r);

    printf("\nMinimum cost of MST is: %d",minCost);

    return 0;

}

**Output:**



**Analysis:**

**Time Complexity Analysis:**

• The time complexity of Kruskal's algorithm using Union-Find data structure is O(E log E) or O(E log V), where E is the number of edges in the graph and V is the number of vertices in the graph.

• The complexity of sorting the edges using a sorting algorithm such as quicksort or merge sort is O(E log E) or O(E log V). The Union-Find operations take O(log V) time in the worst case, and the algorithm performs E such operations. Therefore, the total time complexity of Kruskal's algorithm is O(E log E + E log V) or O(E log V).

• In the worst case, the number of edges in a dense graph is O(V^2), which makes the time complexity O(V^2 log V) or O(V^3 log V). However, in practice, sparse graphs are more common, and the time complexity is much closer to O(E log V).