**Third Year B. Tech., Sem V 2022-23**

**Design and Analysis of Algorithm Lab**

**Lab ESE Assignment / Journal submission**

**PRN/ Roll No: 21520010**

**Full name: Mayur Sunil Savale**

**Batch: T8**

**Title of assignment: kruskal’s Algorithm**

1. Implement Kruskal’s algorithm & Prim’s algorithm to find Minimum Spanning Tree (MST) of the given an undirected, connected and weighted graph.

Ans: **Kruskal’s Algorithm**

a) Algorithm: (Pseudocode) //Initialize result mst\_weight = 0 // Create V single item sets for each vertex v parent[v] = v; rank[v] = 0; Sort all edges into non decreasing order by weight w for each (u, v) taken from the sorted list E do if FIND-SET(u) != FIND-SET(v) print edge(u, v) mst\_weight += weight of edge(u, v) UNION(u, v)

1. **Code snapshots of implementation**

#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;

cout<<"Edge\tWeight\n";

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 << char(u+64) << " - " << char(v+64) <<"\t"<< it->first << endl;

mst\_wt += it->first;

ds.merge(set\_u, set\_v);

}

}

return mst\_wt;

}

int main()

{

int V = 10, E = 19;

Graph g(V, E);

g.addEdge(1,2,-3);

g.addEdge(1,3,1);

g.addEdge(1,4,4);

g.addEdge(2,1,-3);

g.addEdge(2,5,3);

g.addEdge(2,4,5);

g.addEdge(3,4,5);

g.addEdge(3,1,1);

g.addEdge(3,6,3);

g.addEdge(4,1,4);

g.addEdge(4,2,5);

g.addEdge(4,3,5);

g.addEdge(4,6,6);

g.addEdge(5,2,3);

g.addEdge(5,6,2);

g.addEdge(6,3,3);

g.addEdge(6,4,6);

g.addEdge(6,5,2);

cout << "Edges of MST are \n";

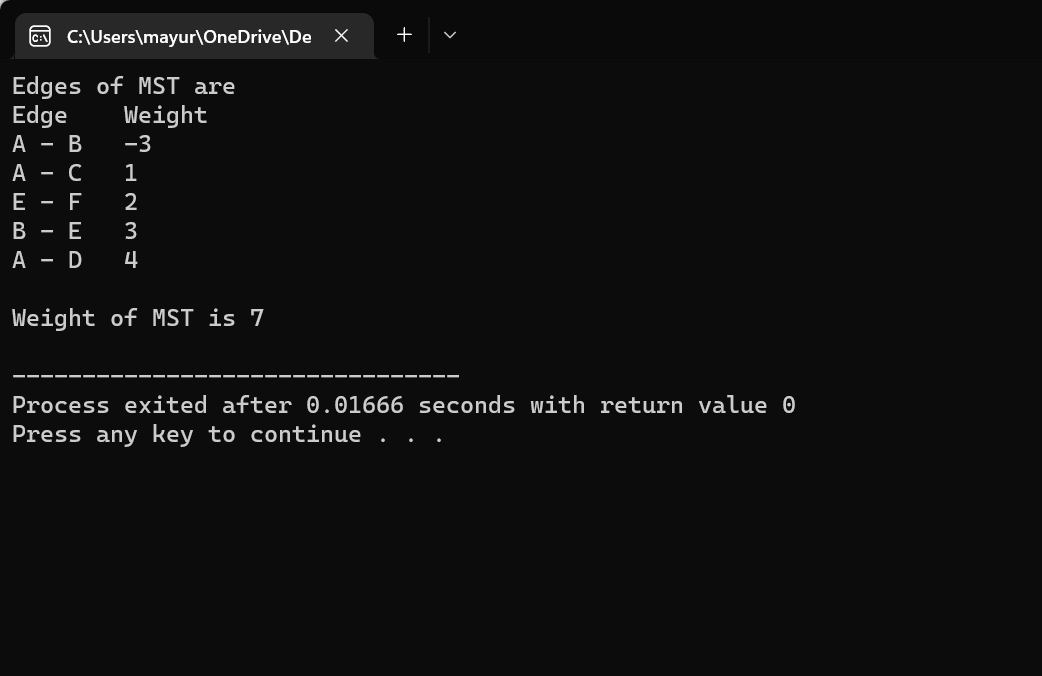
int mst\_wt = g.kruskalMST();

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

return 0;

}

**Output:**



1. **Complexity of proposed algorithm (Time & Space)**

➢ Time Complexity: O(E\*logV)

➢ Space Complexity: O(V+E) d)

**Your comment (How your solution is optimal?)**

Code can be optimized to stop the main loop of Kruskal when number of selected edges becomes V-1