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

Design and Analysis of Algorithm Lab

Assignment / Journal submission

PRN/ Roll No: 21520010

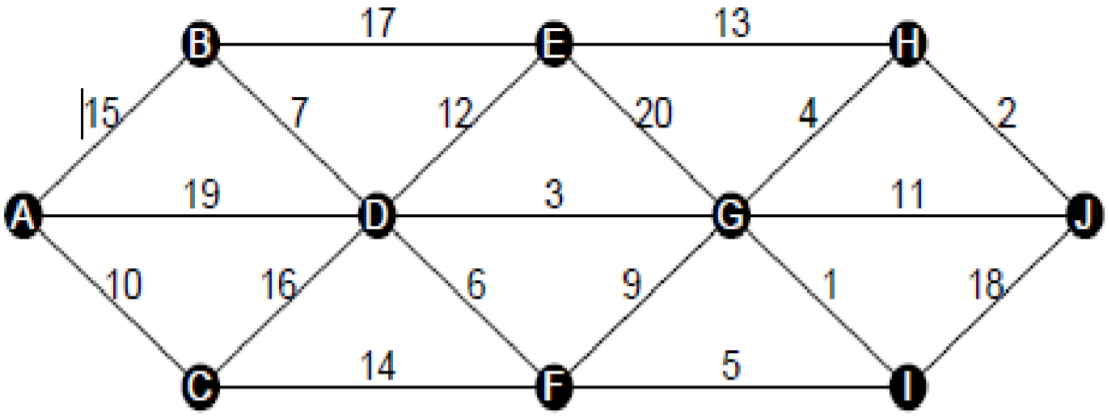
Full name: Mayur Sunil Savale

Batch: T8

Assignment: Week 7

Title of assignment: Greedy Method

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)
```

b) 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+65) << " - " << char(v+65) <<"\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(0, 1, 15);
        g.addEdge(0, 2, 10);
        g.addEdge(0, 3, 19);
        g.addEdge(1, 3, 7);
        g.addEdge(1, 4, 17);
        g.addEdge(2, 3, 16);
        g.addEdge(2, 5, 14);
        g.addEdge(3, 4, 12);
        g.addEdge(3, 5, 6);
        g.addEdge(3, 6, 3);
        g.addEdge(4, 6, 20);
        g.addEdge(4, 7, 13);
        g.addEdge(5, 6, 9);
        g.addEdge(5, 8, 5);
        g.addEdge(6, 7, 4);
        g.addEdge(6, 8, 1);
        g.addEdge(6, 9, 11);
```

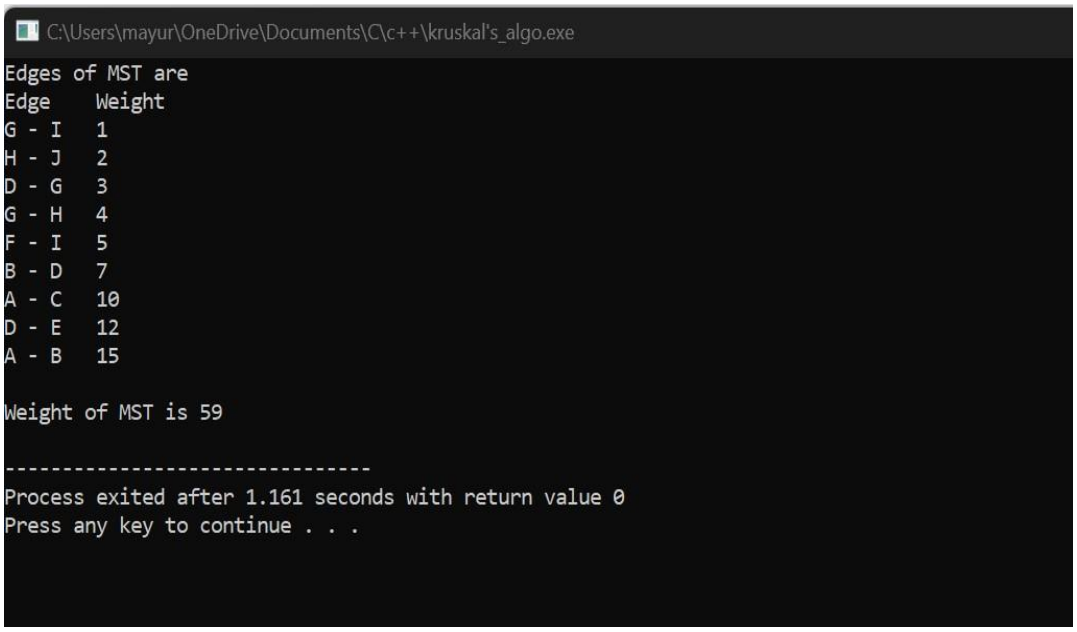
```
        g.addEdge(7, 9, 2);
        g.addEdge(8, 9, 18);

        cout << "Edges of MST are \n";
        int mst_wt = g.kruskalMST();

        cout << "\nWeight of MST is " << mst_wt << "\n";

        return 0;
}
```

Output:



c) 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

Prim's Algorithm

a) Algorithm: (Pseudocode)

- Initialize keys of all vertices as infinite and parent of every vertex as -1
- Create an empty priority_queue pq. Every item of pq is a pair (weight, vertex) Weight (or key) is used used as first item of pair as first item is by default used to compare two pairs.
- Initialize all vertices as not part of MST yet. We use boolean array inMST[] for this purpose. This array is required to make sure that an already considered vertex is not included in pq again. This is where Prim's implementation differs from Dijkstra.
- Insert source vertex into pq and make its key as 0.
- While either pq doesn't become empty
    a. Extract minimum key vertex from pq. Let the extracted vertex be u.
    b. Include u in MST using inMST[u] = true.
    c. Loop through all adjacent of u and do following for every vertex v.

        // If weight of edge (u,v) is smaller than key of v and v is not already in MST
        If inMST[v] = false && key[v] > weight(u, v)
            (i) Update key of v, i.e., do
            key[v] = weight(u, v)
        (ii) Insert v into the pq
        (iv) parent[v] = u

    d. Print MST edges using parent array.

b) Code snapshots of implementation

```
#include<bits/stdc++.h>
using namespace std;
# define INF 0x3f3f3f3f

typedef pair<int, int> iPair;
```

```
void addEdge(vector <pair<int, int> > adj[], int u,
                                                    int v, int wt)
{
        adj[u].push_back(make_pair(v, wt));
        adj[v].push_back(make_pair(u, wt));
}

void primMST(vector<pair<int,int> > adj[], int V)
{
        priority_queue< iPair, vector <iPair> , greater<iPair> > pq;

        int src = 0;
        vector<int> key(V, INF);

        vector<int> parent(V, -1);

        vector<bool> inMST(V, false);

        pq.push(make_pair(0, src));
        key[src] = 0;

        while (!pq.empty())
        {

                int u = pq.top().second;
                pq.pop();

                if(inMST[u] == true){
                        continue;
                }

                inMST[u] = true;

                for (auto x : adj[u])
```

```
                {

                        int v = x.first;
                        int weight = x.second;

                        if (inMST[v] == false && key[v] > weight)
                        {
                                // Updating key of v
                                key[v] = weight;
                                pq.push(make_pair(key[v], v));
                                parent[v] = u;
                        }
                }
        }
        cout<<"Edges\tWeight\n";
        for (int i = 1; i < V; ++i)
                cout << char(parent[i]+65) << " - " << char(i+65)
<<"\t"<<key[i]<< "\n";
}


int main()
{
        int V = 10;
        vector<iPair > adj[V];

        addEdge(adj, 0, 1, 15);
        addEdge(adj, 0, 2, 10);
        addEdge(adj, 0, 3, 19);
        addEdge(adj, 1, 3, 7);
        addEdge(adj, 1, 4, 17);
        addEdge(adj, 2, 3, 16);
        addEdge(adj, 2, 5, 14);
        addEdge(adj, 3, 4, 12);
        addEdge(adj, 3, 5, 6);
```

```
    addEdge(adj, 3, 6, 3);
    addEdge(adj, 4, 6, 20);
    addEdge(adj, 4, 7, 13);
    addEdge(adj, 5, 6, 9);
    addEdge(adj, 5, 8, 5);
    addEdge(adj, 6, 7, 4);
    addEdge(adj, 6, 8, 1);
    addEdge(adj, 6, 9, 11);
    addEdge(adj, 7, 9, 2);
    addEdge(adj, 8, 9, 18);

    primMST(adj, V);

    return 0;
}
```

Output:

```
PS C:\Users\mayur> cd "c:\Users\mayur\OneDrive\Desktop\5 th sem\DAA Assignment\Assignment-7\" ; if ($?) { g++ Q2.cpp
Edges of MST are
Edges   Weight
D - B   7
A - C   10
G - D   3
D - E   12
C - F   14
I - G   1
G - H   4
F - I   5
H - J   2
PS C:\Users\mayur\OneDrive\Desktop\5 th sem\DAA Assignment\Assignment-7>
PS C:\Users\mayur\OneDrive\Desktop\5 th sem\DAA Assignment\Assignment-7>
PS C:\Users\mayur\OneDrive\Desktop\5 th sem\DAA Assignment\Assignment-7>
PS C:\Users\mayur\OneDrive\Desktop\5 th sem\DAA Assignment\Assignment-7>
```

c) Complexity of proposed algorithm (Time & Space)

- Time Complexity: O(E*logV)
- Space Complexity: O(V+E)

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

- We achieved time complexity of O(E*logV) using priority queue. As implementation using adjacency matrix gives time complexity of O(V*2)

2. How many edges does a minimum spanning tree for above example?

Ans:

As there are total 10 vertices so the number of edges in minimum spanning tree are 9.

3. In a graph G let the edge u, v has the least weight is it true that u, v is always part of any minimum spanning tree of G? Justify your answers.

Ans:

Yes (u, v) always part of any minimum spanning tree of G.

To prove this,

Assume that none of the MST contain the edge (u,v).

Pick a minimum spanning tree S. It will contain a path from u to V. Pick any edge i.e. (w, z) in that path and remove it. This will create two disconnected components one contains u and other contains v. Connect these two components v with edge (u, v). This new graph is a spanning tree S'.

Weight of S – Weight of S' = weight of (w, z) -weight of (u, v) >=0

Therefore, weight of S >= weight of S'

Hence proved.

4. Let G be a graph and T be a minimum spanning tree of G. Suppose that the weight of an edge e is decreased. How can you find the minimum spanning tree of the modified graph? What is the runtime of your solution?

Ans:

First of all, we need to look if the decreased weight of edge makes any impact on minimum spanning tree, i.e. if the decreased weight is less

than last edge of tree, we need to sort the edge array again and re-construct the minimum spanning tree.
If such scenario does not occurs then we don't need to reconstruct the tree.

Time Complexity: Worst Case: O(E*logV)
Best Case: O(E)

5. Find order of edges for Kruskal's and Prim's?

Ans:

Order of edges are:

Prim's Algorithm:

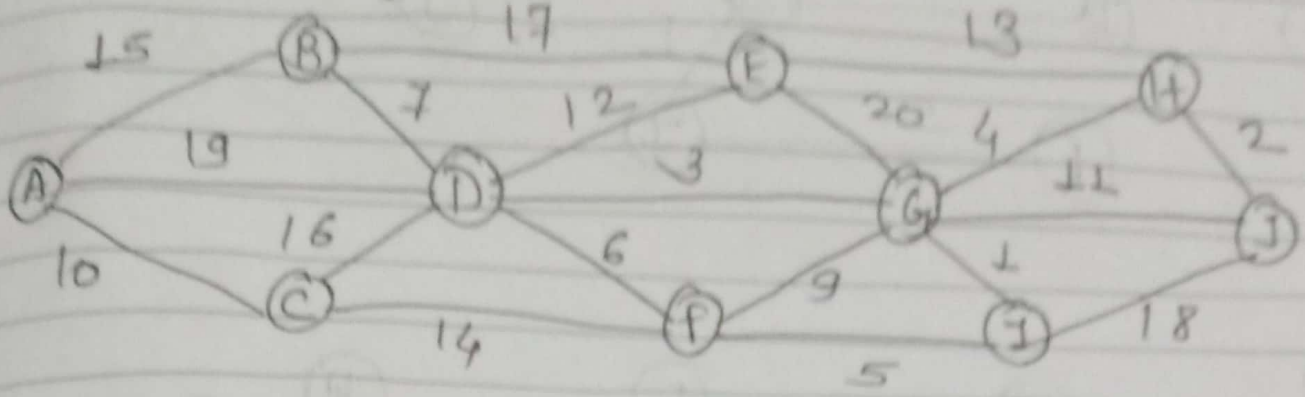
D - B, A - C, G - D, D - E, C - F, I - G, G - H, F - I, H - J

Kruskal's Algorithm:

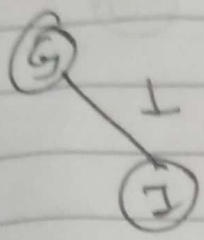
G – I, H – J, D- G, G- H, F- I, B- D, A- C, D-E, C- F

Kruskal's algorithm:

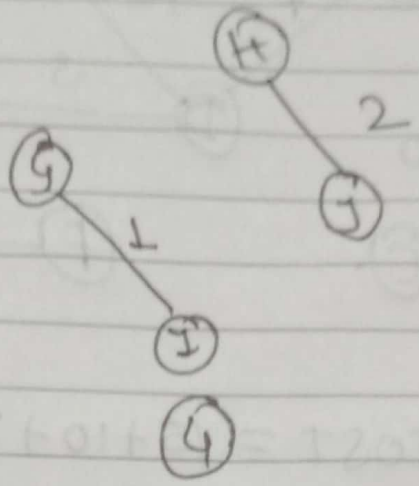
sol)



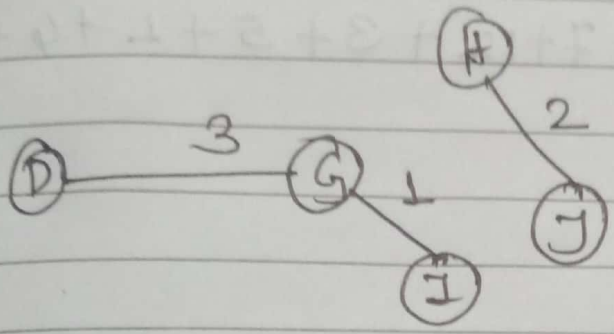
ans: 1)

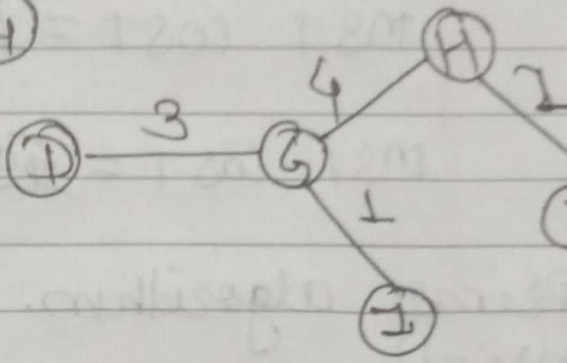


2)

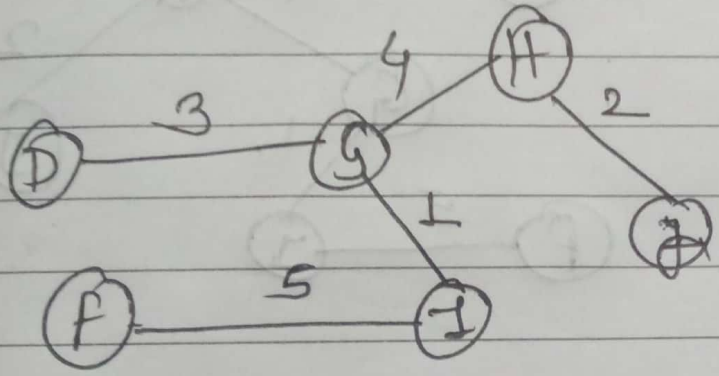


3)

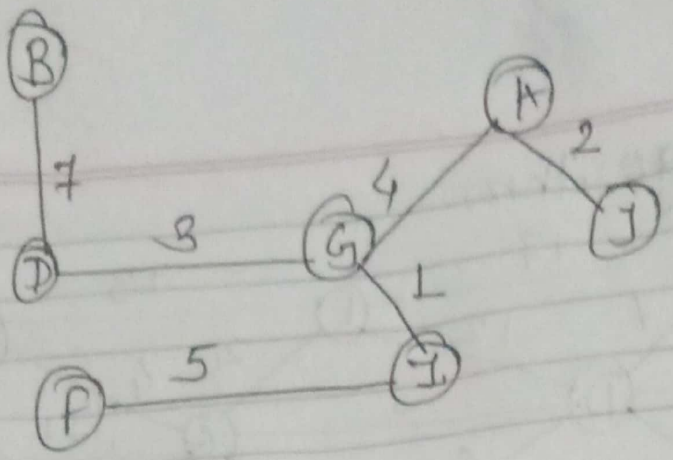




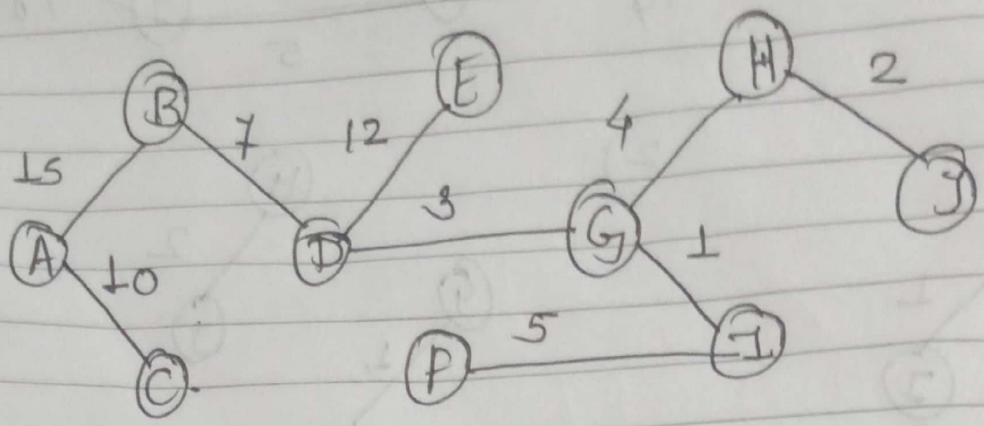
5)



(6)



(7)

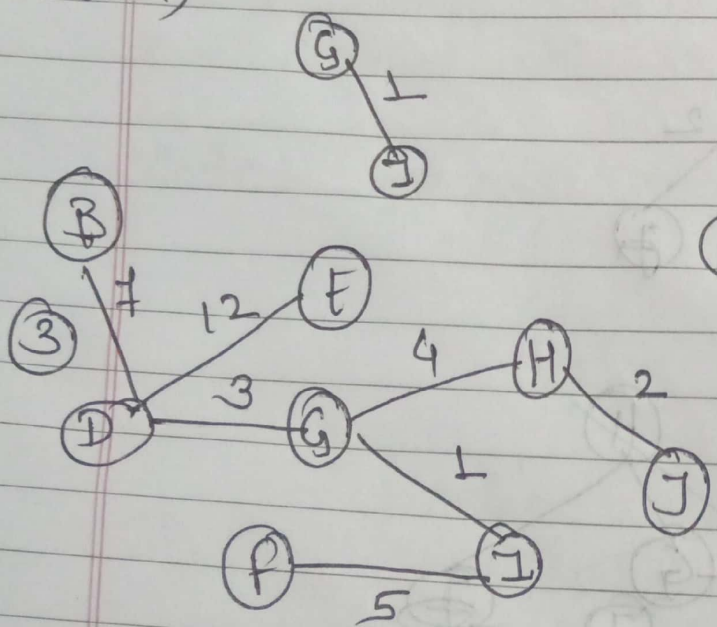


MST COST = 15+10+7+12+3+5+1+4+2

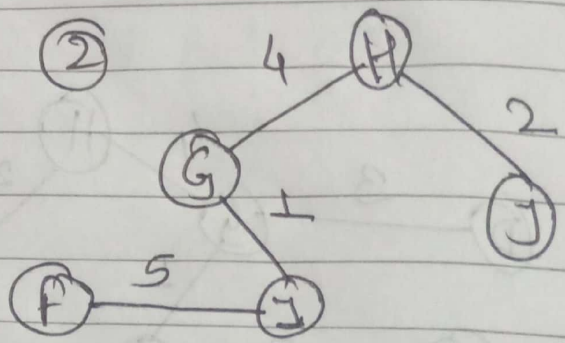
MST COST = 59.

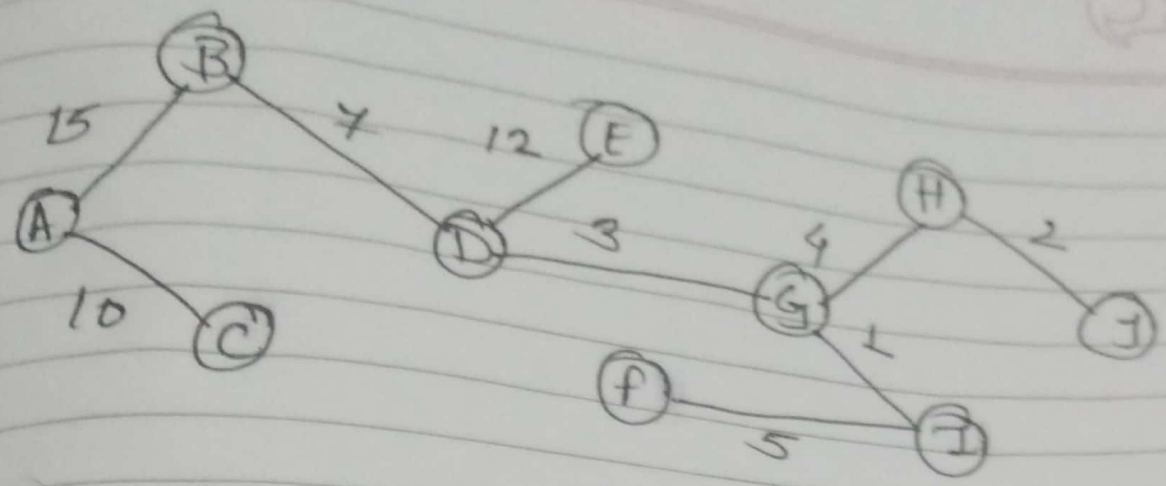
Prims algorithm.

ans:-1)



(2)





MST Cost = 15 + 10 + 7 + 12 + 3 + 5 + 4 + 1 + 2

= 59