

## EXPERIMENT 6

**Experiment No:** 6

**Date:** 10/04/2021

**Aim:** Implementation of Minimum Cost Spanning Tree(Using Prim's Algo) and estimate its step count

**Theory:**

### **PRIM'S ALGORITHM**

- Prim's algorithm is also a Greedy algorithm.
- It starts with an empty spanning tree.
- The idea is to maintain two sets of vertices.
- The first set contains the vertices already included in the MST.
- The other set contains the vertices not yet included.
- At every step, it considers all the edges that connect the two sets, and picks the minimum weight edge from these edges.
- After picking the edge, it moves the other endpoint of the edge to the set containing MST.

### **HOW PRIM'S ALGORITHM WORK**

- It falls under a class of algorithms called greedy algorithms that find the local optimum in the hopes of finding a global optimum.
- We start from one vertex and keep adding edges with the lowest weight until we reach our goal.
- The steps for implementing Prim's algorithm are as follows:
  - Initialize the minimum spanning tree with a vertex chosen at random.

## EXPERIMENT 6

- Find all the edges that connect the tree to new vertices, find the minimum and add it to the tree
- Keep repeating step 2 until we get a minimum spanning tree.

### **ALGORITHM**

```
Algorithm prim(v,n,parent,cost)
{
    //v is graph
    //n is no of vertices
    //let cost be the array of minimum cost for each vertex to
    become a part of spanning tree
    //let parent be the array which will store the immediate
    parent of each vertex
    // initialize total cost to 0
    total := 0
    no_of_edges=0
    cost[1] = 0 //cost of starting vertex is 0
    while(no_of_edges!=n-1)
    {
        //find n-1 edges in graph to create a spanning tree
        Let x be an unvisited vertex with minimum cost
        among all other unvisited vertices to connect to
        spanning tree
        //To find x we need to run throust cost array with min
```

## EXPERIMENT 6

variable to track minimum cost

min := Inf

for i:=1 to n do

{

if(visited[i]==false and min>cost[i]) then

{

min := cost[i];

x := i

}

}

visited[x] = true;

for i:=1 to n do

{

if(visited[i]==false and v[x][i]<cost[i]) then

{

cost[i] := v[x][i]

parent[i] := x

}

}

no\_of\_edges++;

}

}

## EXPERIMENT 6

### **ALGORITHM**

- Create a set `mstSet` that keeps track of vertices already included in MST.
- Assign a key value to all vertices in the input graph. Initialize all key values as INFINITE.
- Assign key value as 0 for the first vertex so that it is picked first.
- While `mstSet` doesn't include all vertices
  - Pick a vertex `u` which is not there in `mstSet` and has minimum key value.
  - Include `u` to `mstSet`.
  - Update key value of all adjacent vertices of `u`.
  - To update the key values, iterate through all adjacent vertices.
  - For every adjacent vertex `v`, if weight of edge `u-v` is less than the previous key value of `v`, update the key value as weight of `u-v`
- The idea of using key values is to pick the minimum weight edge from cut. The key values are used only for vertices which are not yet included in MST, the key value for these vertices indicate the minimum weight edges connecting them to the set of vertices included in MST.

## EXPERIMENT 6

### TIME COMPLEXITY

- If adjacency list is used to represent the graph, then using breadth first search, all the vertices can be traversed in  $O(V + E)$  time.
- We traverse all the vertices of graph using breadth first search and use a min heap for storing the vertices not yet included in the MST.
- To get the minimum weight edge, we use min heap as a priority queue.
- Min heap operations like extracting minimum element and decreasing key value takes  $O(\log V)$  time.

***So, overall time complexity***

$$= O(E + V) \times O(\log V)$$

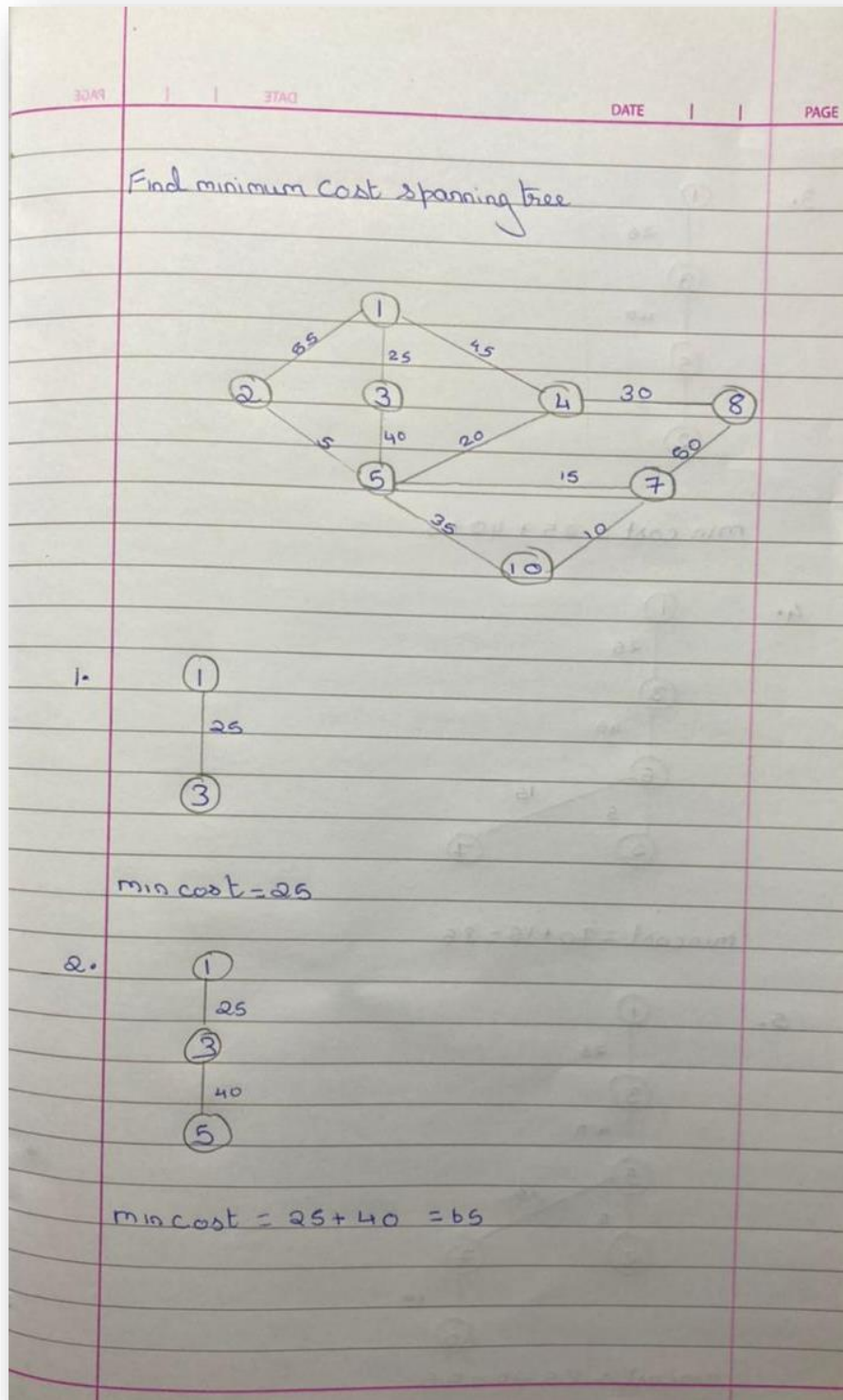
$$= O((E + V)\log V)$$

$$= O(E\log V)$$

- This time complexity can be improved and reduced to  $O(E + V\log V)$  using Fibonacci heap.

## EXPERIMENT 6

### TRACING WITH EXAMPLE



## EXPERIMENT 6

3.

```
graph TD; 1((1)) ---|25| 3((3)); 3 ---|40| 5((5)); 5 ---|5| 2((2))
```

$\text{min cost} = 25 + 40 + 5$

4.

```
graph TD; 1((1)) ---|25| 3((3)); 3 ---|40| 5((5)); 5 ---|5| 2((2)); 5 ---|15| 7((7))
```

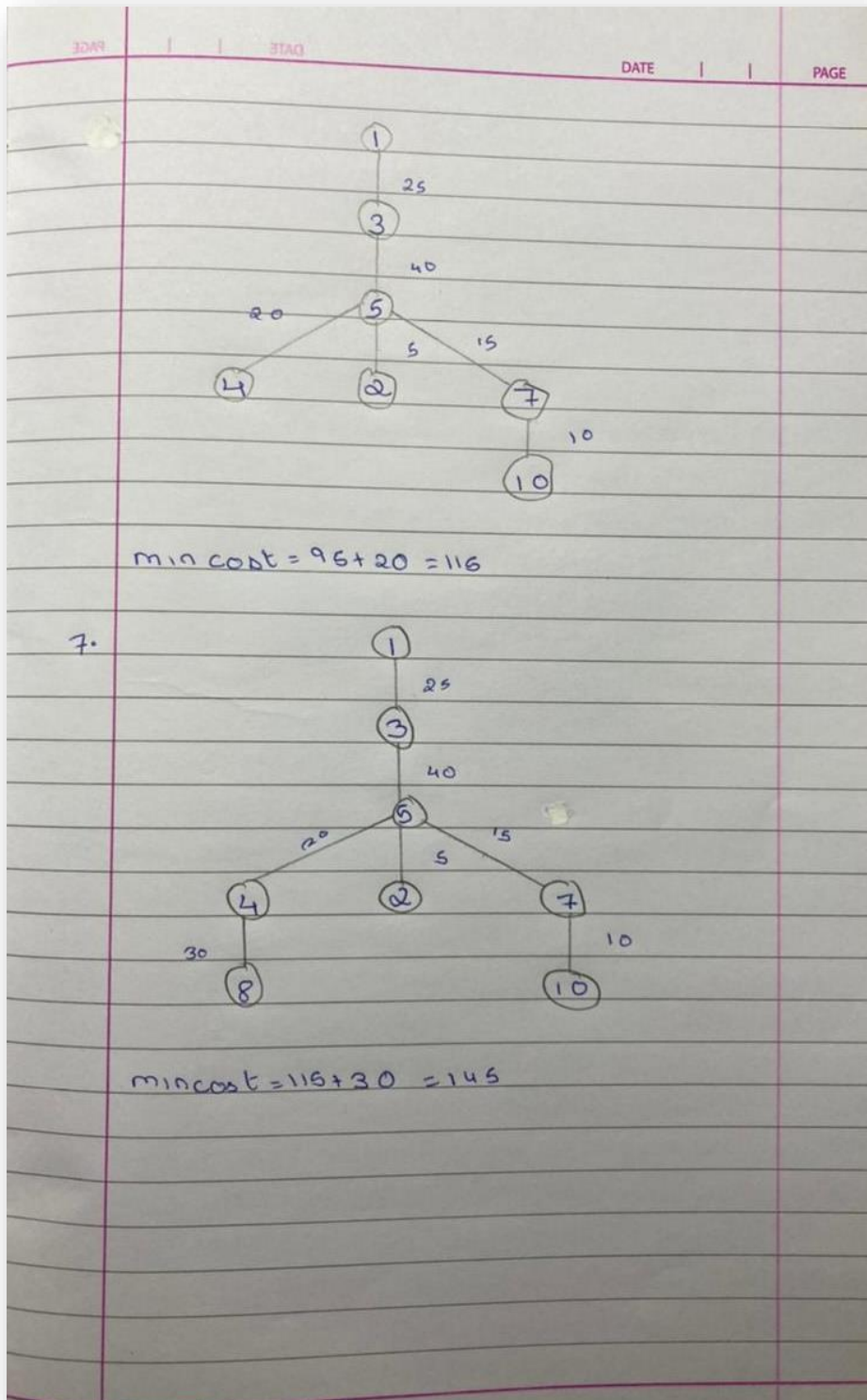
$\text{min cost} = 70 + 15 = 85$

5.

```
graph TD; 1((1)) ---|25| 3((3)); 3 ---|40| 5((5)); 5 ---|5| 2((2)); 5 ---|15| 7((7)); 7 ---|10| 10((10))
```

$\text{min cost} = 85 + 10 = 95$

## EXPERIMENT 6





## EXPERIMENT 6

### **PROGRAM**

```
#include<bits/stdc++.h>

using namespace std;

int c=0;

void primsAlgo(    vector<vector<pair<int,int>>> v,int n)
{
    bool visited[n+1];
    int parent[n+1];
    int cost[n+1];
    memset(visited,false,sizeof(visited));
    memset(parent,-1,sizeof(parent));


    for(int i=1;i<=n;i++) {
        c++; //for for loop
        cost[i] = INT_MAX;
        c++;//for assignment
    }
    c++;//for last for loop


    c++;//for assigning cost 0 to first vertex
    cost[1] = 0;


    int total=0;
```

## EXPERIMENT 6

```
int edges = 0;
while(edges!=n-1)
{
    c++;//for while loop
int x;
int min=INT_MAX;
for(int i=1;i<=n;i++)
{
    c++;//for loop
    c++;//for if
    if(visited[i]==false&&cost[i]<min)
    {

        min = cost[i];
        c++;//for assignment
        x=i;
        c++;//for assignment
    }
}
c++;//for last for loop
visited[x] = true;
c++;//for assigning true value

for(auto i:v[x])
{
```

## EXPERIMENT 6

```
c++; // for loop
c++; // for if
if(visited[i.first]==false && i.second < cost[i.first])
{
    cost[i.first] = i.second;
    c++; // for assignment
    parent[i.first] = x;
    c++; // for assignment
}
}
c++; // for last for loop

edges++;
c++; // for incrementing

}
c++; // for last while loop
for(int i=2; i<=n; i++)
{
    c++; // if condition
    if(cost[i] != INT_MAX)
    {
        cout<<i<<" < - > "<<parent[i]<<" cost:
        "<<cost[i]<<endl<<endl;
        c++; // for cout
        total+=cost[i];
```

## EXPERIMENT 6

```
        c++;
    }
}

cout<<"*****"<<endl;
cout<<"Total Cost: "<<total<<endl;
cout<<"*****"<<endl;
c++;
}

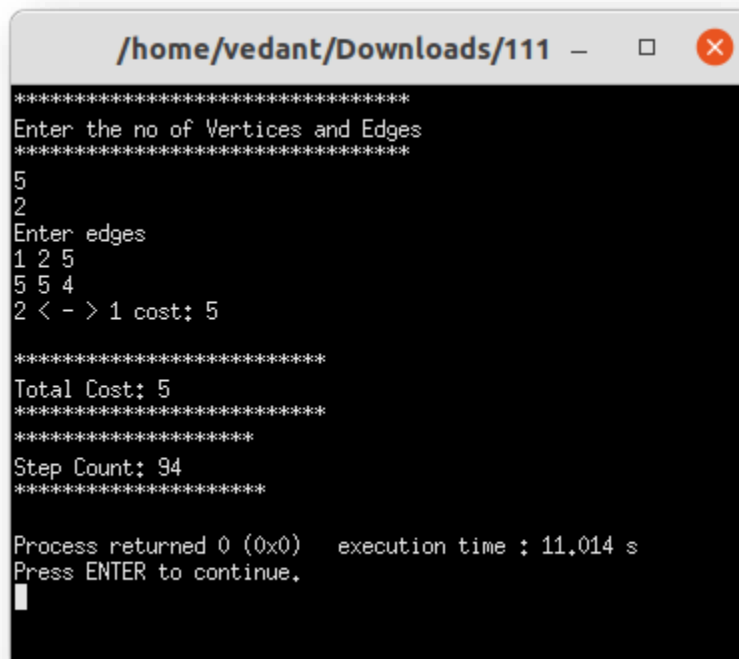
int main()
{
    int n,m;
    cout<<"*****"<<endl;
    cout<<"Enter the no of Vertices and Edges"<<endl;
    cout<<"*****"<<endl;
    cin>>n>>m;
    vector<vector<pair<int,int>>> v(n+1);
    cout<<"Enter edges"<<endl;
    for(int j=0;j<m;j++)
    {

        int x,y,w;
        cin>>x>>y>>w;
        v[x].push_back({y,w});
        v[y].push_back({x,w});
    }
```

## EXPERIMENT 6

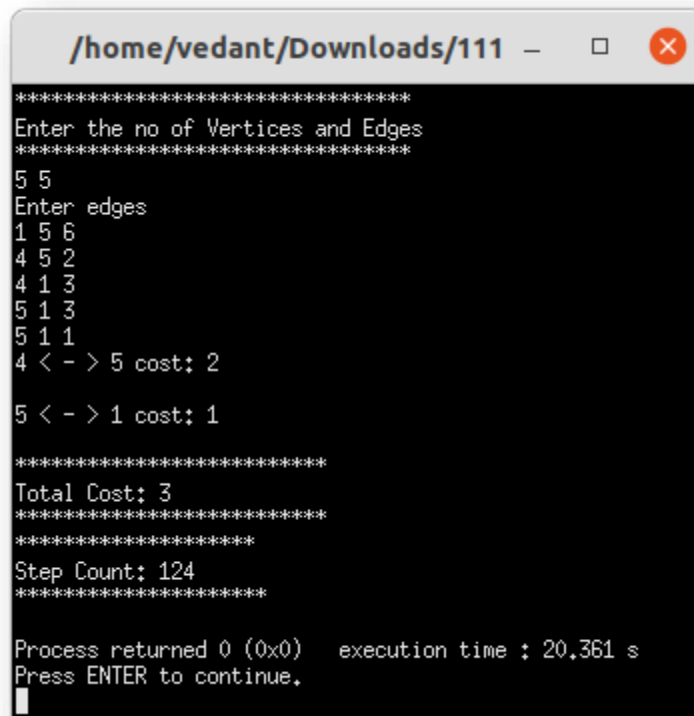
```
primsAlgo(v,n);  
  
cout<<"*****"<<endl;  
  
cout<<"Step Count: "<<c<<endl;  
  
cout<<"*****"<<endl;  
  
}
```

### OUTPUT



```
*****  
Enter the no of Vertices and Edges  
*****  
5  
2  
Enter edges  
1 2 5  
5 5 4  
2 < - > 1 cost: 5  
  
*****  
Total Cost: 5  
*****  
*****  
Step Count: 94  
*****  
  
Process returned 0 (0x0)   execution time : 11.014 s  
Press ENTER to continue.  
|
```

## EXPERIMENT 6



```
/home/vedant/Downloads/111 - [X]
*****
Enter the no of Vertices and Edges
*****
5 5
Enter edges
1 5 6
4 5 2
4 1 3
5 1 3
5 1 1
4 < - > 5 cost: 2
5 < - > 1 cost: 1
*****
Total Cost: 3
*****
Step Count: 124
*****
Process returned 0 (0x0)   execution time : 20.361 s
Press ENTER to continue.
```

### CONCLUSION

- Detailed concept of Minimum Cost Spanning Tree(Using Prim's Algo) was studied successfully.
- Program using Prim's Algorithm was executed successfully.
- The step count for the Prim's algorithm was obtained