**Experiment No: 6**

**AIM:** Implementation Minimum Cost Spanning tree(using Prim’s Algorithm) and obtaining its step count.

**THEORY:**

Prim’s algorithm is 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.

A group of edges that connects two set of vertices in a graph is called cut in graph theory. So, at every step of Prim’s algorithm, we find a cut (of two sets, one contains the vertices already included in MST and other contains rest of the vertices), pick the minimum weight edge from the cut and include this vertex to MST Set (the set that contains already included vertices).

*How does Prim’s Algorithm Work?*

The idea behind Prim’s algorithm is simple, a spanning tree means all vertices must be connected. So, the two disjoint subsets (discussed above) of vertices must be connected to make a Spanning Tree. And they must be connected with the minimum weight edge to make it a Minimum Spanning Tree.

**Algorithm writing**

* In this algorithm, we take E as the set of edges in G (Undirected Graph).
* Cost [1:n, 1:n] is the cost adjacency matrix of an n vertex graph such that cost[i,j] is either a positive real number or infinity if no edge (i,j) exists.
* A minimum spanning tree is computed and stored as a set of edges in the array t[1:n-1, 1:2]. (t[i,1], t[i,2]) is an edge in the minimum-cost spanning tree.
* The final cost is returned.

**ALGORITHM:**

**Algorithm** Prim (E, cost, n, t)

{

//Let (k,l) be an edge of minimum cost in E;

mincost := cost[k,l];

t[1,1] := k; t[1,2] := l;

**for** i:=1 **to** n **do** //Initialise near.

**if**(cost[i,l] < cost[i,k]) **then** near[i] := l;

**else** near[i] = k;

near[k] := near[l] := 0;

**for** i:= 2 **to** n-1 **do**

{

//Find n-2 additional edges for t.

//Let j be an index such that near[j] != 0 and cost[j, near[j]] is minimum;

t[i,1] := j; t[i,2] := near[j];

mincost := mincost + cost[j, near[j]];

near[j] := 0;

**for** k:=1 **to** n **do** //Update near[].

**if**(near[k]!=0) **and** (cost[k, near[k]] > cost[k,j])

**then** near[k] := j;

}

**return** mincost;

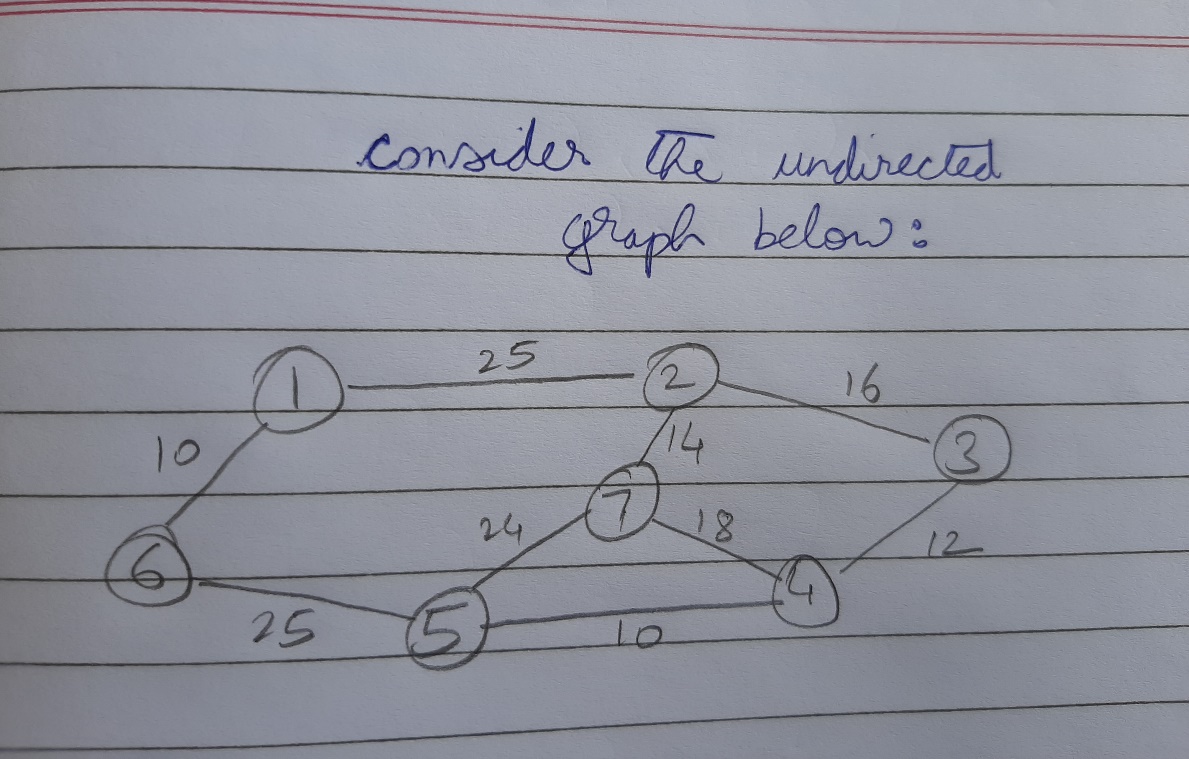
}

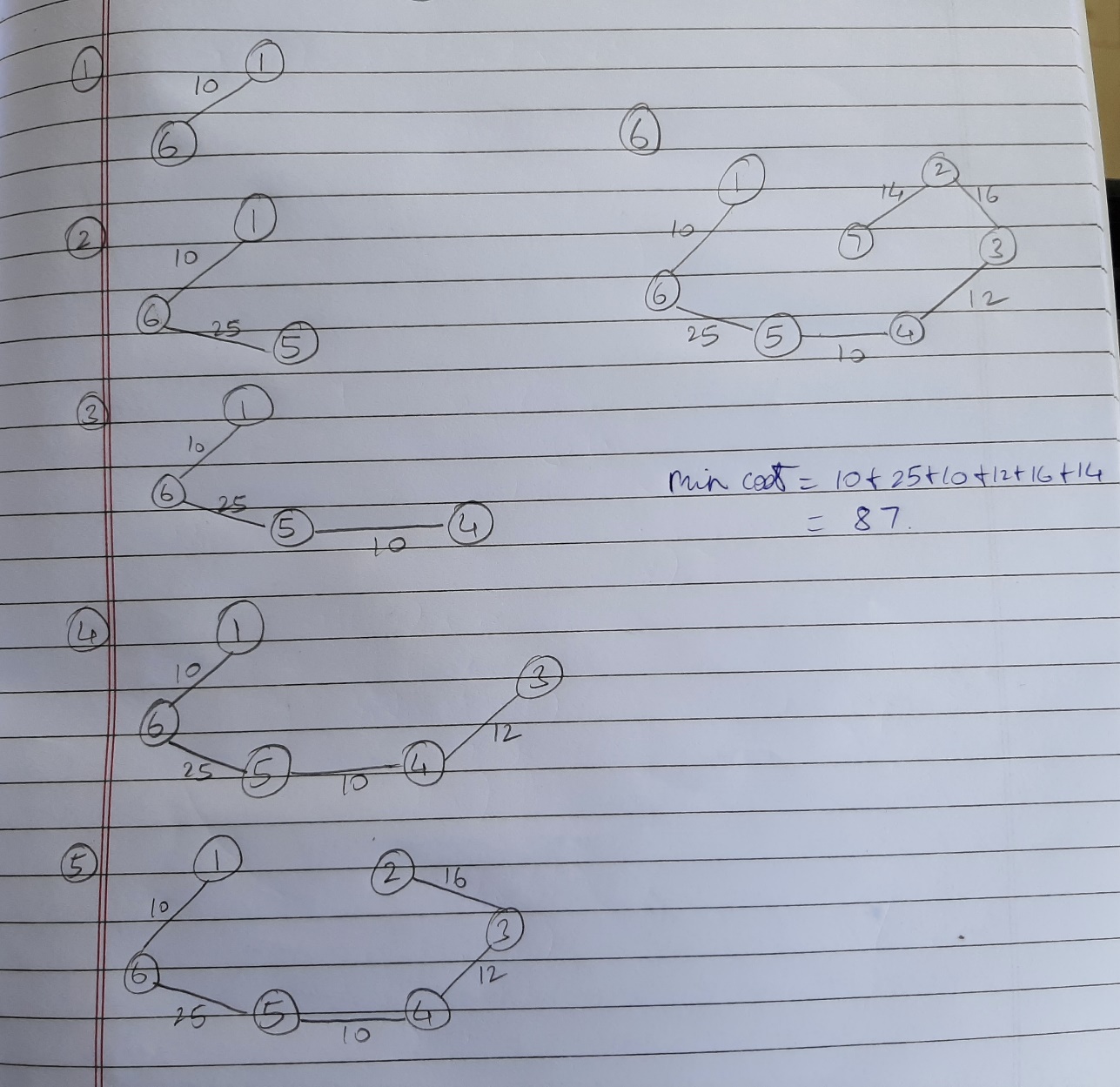
*Time Complexity*

• Time Complexity of the above program is O(V^2).

• If the input graph is represented using adjacency list, then the time complexity of Prim’s algorithm can be reduced to O(E log V) with the help of binary heap.

*Problem Tracing*





PROGRAM IMPLEMENTATION:

#include<iostream>

using namespace std;

const int v=10; //size of the matrix

int count=0,n; //n is number of vertices

void showtree(int \*parent, int graph[v][v])

{

cout<<"The minimum cost spanning tree is:\n";

cout<<"Edge Weight\n";

for(int i=1;i<n;i++)

{

cout<<parent[i]<<" - "<<i<<" \t"<<graph[i][parent[i]]<<"\n";

count++; //for

}

count++; //for

int minkey(int \*key,bool \*visited)

{

int min=99,min\_index; //99 indicates infinite value

count++; //for assign

for(int i=0;i<n;i++)

{

count+=2; //for and if

if(key[i]<min && visited[i]==false)

{

min=key[i];

min\_index=i;

count+=2;

}

}

count++; //for

count++; //return

return min\_index;

}

void mintree(int graph[v][v])

{

int parent[n],key[n]; //parent indicates the spanning tree; key indicates the weights;

bool visited[n]; //visited indicates whether the vertice has been visited or not

for(int i=0;i<n;i++)

{

key[i]=99,visited[i]=false;

count+=3;

}

count++;

key[0]=0; //first vertex is chosen first

parent[0]=-1; //root of mst

count+=2;

for(int i=0;i<n-1;i++)

{

int u = minkey(key,visited);

visited[u] = true;

count+=3; //outer for and assignments

for(int j=0;j<n;j++)

{

count+=2; //inner for and if

if(graph[u][j] && visited[j]==false && graph[u][j]<key[j])

{

key[j]=graph[u][j],

parent[j]=u;

count+=2;

}

}

count++; //inner for

}

count++; //outer for

showtree(parent,graph);

}

int main()

{

int graph[v][v];

cout<<"Enter number of vertices:";

cin>>n;

for(int i=0;i<n;i++)

for(int j=0;j<n;j++)

{

cout<<"\nEnter edge cost between "<<i<<" and "<<j<<":";

cin>>graph[i][j];

}

mintree(graph);

cout<<"\nCount="<<count<<endl;

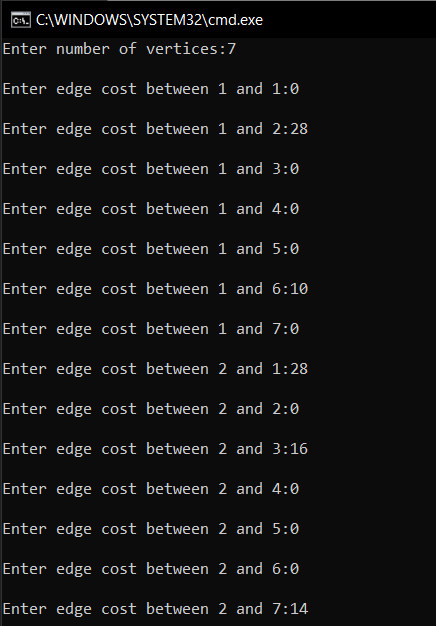
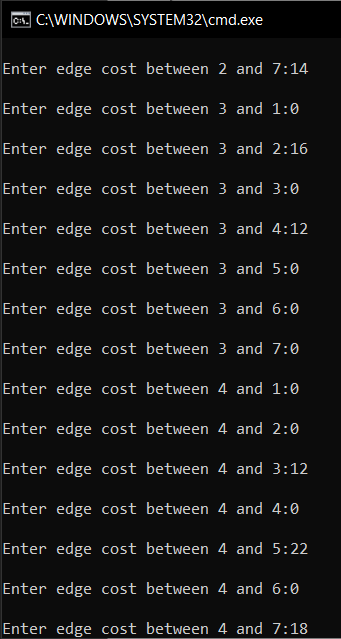
return 0;

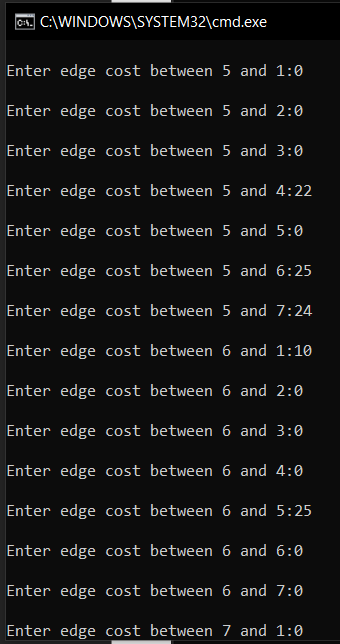
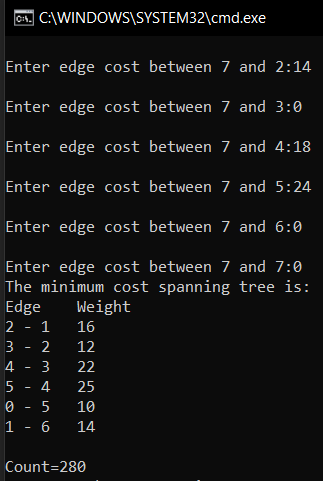
}

OUTPUTS:

1. When n=7 (number of vertices)

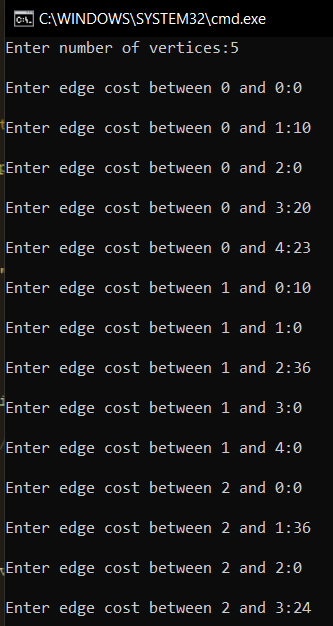
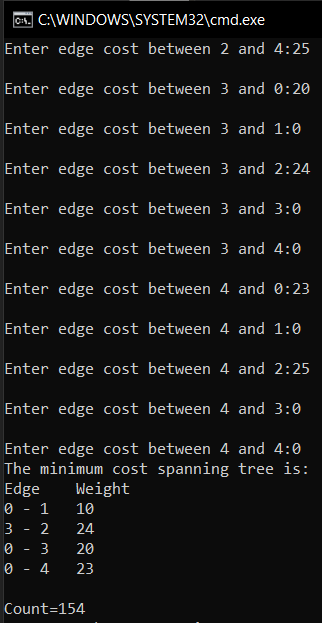
**Count=280**

1. When n=5 (number of vertices)

**Count=154**

** **

**Conclusion**:

1. **Prim’s algorithm to compute minimum spanning tree takes O(**) **time, where V is the number of vertices in the graph.**
2. **If a graph is represented by its adjacency list, then the time taken by the algorithm reduces to O(E logV)**