BFS :

```
#include<iostream>

#include<stdlib.h>
using namespace std;
int cost[10][10],i,j,k,n,qu[10],front,rare,v,visit[10],visited[10];
int main()
{
    int m;
    cout <<"Enter no of vertices:";
    cin >> n;
    cout <<"Enter no of edges:";
    cin >> m;
    cout <<"\nEDGES \n";
    for(k=1; k<=m; k++)
    {
        cin >>i>>j;
        cost[i][j]=1;
    }
    cout <<"Enter initial vertex to traverse from:";
    cin >>v;
    cout <<"Visitied vertices:";
    cout <<v<<" ";
    visited[v]=1;
    k=1;
    while(k<n)
    {
        for(j=1; j<=n; j++)
```

```
            if(cost[v][j]!=0 && visited[j]!=1 && visit[j]!=1)
            {
                visit[j]=1;
                qu[rare++]=j;
            }
        v=qu[front++];
        cout<<v <<" ";
        k++;
        visit[v]=0;
        visited[v]=1;
    }
    return 0;
}
```

DFS :

```
#include<iostream>
#include<conio.h>
#include<stdlib.h>
int cost[10][10],i,j,k,n,stk[10],top,v,visit[10],visited[10];
int main()
{
    int m;
    cout <<"Enter no of vertices:";
    cin >> n;
    cout <<"Enter no of edges:";
    cin >> m;
    cout <<"\nEDGES \n";
    for(k=1; k<=m; k++)
    {
        cin >>i>>j;
        cost[i][j]=1;
    }
    cout <<"Enter initial vertex to traverse from:";
    cin >>v;
    cout <<"DFS ORDER OF VISITED VERTICES:";
    cout << v <<" ";
    visited[v]=1;
    k=1;
    while(k<n)
    {
        for(j=n; j>=1; j--)
            if(cost[v][j]!=0 && visited[j]!=1 && visit[j]!=1)
            {
                visit[j]=1;
                stk[top]=j;
                top++;
            }
        v=stk[--top];
        cout<<v << " ";
        k++;
        visit[v]=0;
        visited[v]=1;
    }
    return 0;
}
```

PRIMS :

```
#include <bits/stdc++.h>
using namespace std;


#define V 7


int minKey(int key[], bool mstSet[])
{

	int min = INT_MAX, min_index;

	for (int v = 0; v < V; v++)
		if (mstSet[v] == false && key[v] < min)
			min = key[v], min_index = v;

	return min_index;
}


void printMST(int parent[], int graph[V][V])
{
	cout << "Edge \tWeight\n";
	for (int i = 1; i < V; i++)
		cout << parent[i] << " - " << i << " \t"
			<< graph[i][parent[i]] << " \n";
}


void primMST(int graph[V][V])
{

	int parent[V];


	int key[V];


	bool mstSet[V];


	for (int i = 0; i < V; i++)
		key[i] = INT_MAX, mstSet[i] = false;


	key[0] = 0;
	parent[0] = -1;
```

```
        for (int count = 0; count < V - 1; count++) {

                int u = minKey(key, mstSet);


                mstSet[u] = true;


                for (int v = 0; v < V; v++)


                        if (graph[u][v] && mstSet[v] == false
                                && graph[u][v] < key[v])
                                parent[v] = u, key[v] = graph[u][v];
        }


        printMST(parent, graph);
}


int main()
{

        int graph[V][V] = {
 {0, 10, 20, 0,0,0,0},
 {10, 0, 0, 28, 0,0,0},
 {20, 0, 0, 0, 24,12,0},
 {0, 28,0, 0, 16,0,22},
 {0, 0, 24, 16, 0,14,0},
 {0, 0, 12, 0, 14,0,18},
 {0, 0, 0, 22, 0,18,0}};


        primMST(graph);

        return 0;
}
```

Kruskhal:

```
#include <iostream>

#include <vector>

#include <utility>
```

```
#include <algorithm>


using namespace std;

const int MAX = 1e4 + 5;

int id[MAX], nodes, edges;

pair <long long, pair<int, int> > p[MAX];


void initialize()

{

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

        id[i] = i;

}


int root(int x)

{
    while(id[x] != x)
    {
        id[x] = id[id[x]];
        x = id[x];
    }
    return x;
}

void union1(int x, int y)
{
    int p = root(x);
    int q = root(y);
    id[p] = id[q];
}

long long kruskal(pair<long long, pair<int, int> > p[])
{
    int x, y;
    long long cost, minimumCost = 0;
    for(int i = 0;i < edges;++i)
    {

        x = p[i].second.first;
        y = p[i].second.second;
        cost = p[i].first;

        if(root(x) != root(y))
        {
            minimumCost += cost;
            union1(x, y);
        }
    }
    return minimumCost;
}

int main()
{
    int x, y;
    long long weight, cost, minimumCost;
    initialize();
    cout<<endl<<"enter the number of nodes: "<<endl;
    cin >> nodes ;

    cout<<"enter the number of edges: "<<endl;
    cin>> edges;
    cout<<endl;
    for(int i = 0;i < edges;++i)
    {
        cout<<"enter node1, node2, weight :  ";
        cin >> x ;
        cin>> y ;
        cin>> weight;
        p[i] = make_pair(weight, make_pair(x, y));
    }

    sort(p, p + edges);
    minimumCost = kruskal(p);
    cout << "minimumCost is: "<<minimumCost << endl;
```

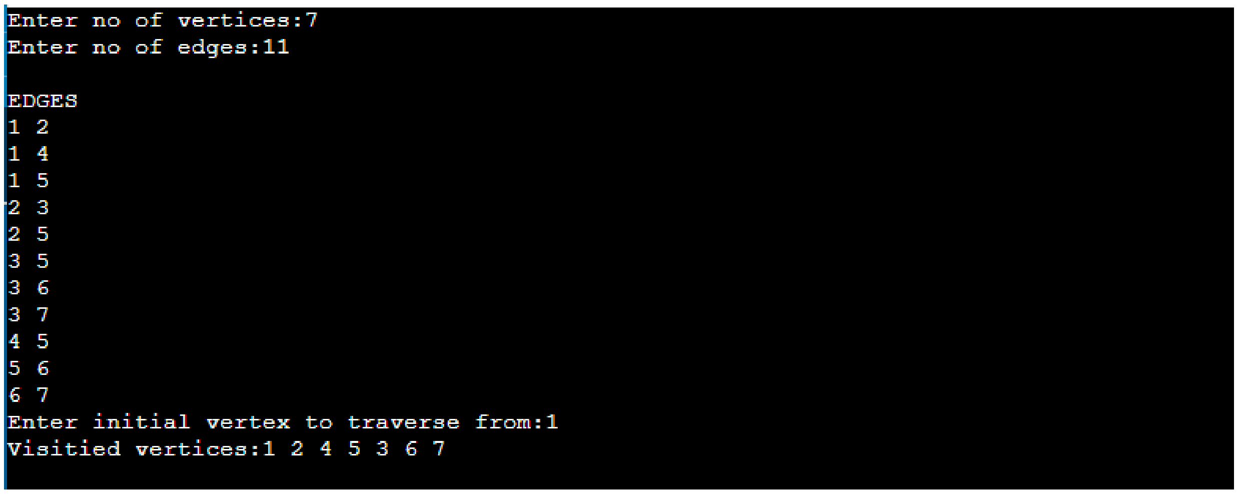
```
return 0;}
```

Dijakstra :

```
#include<bits/stdc++.h>
using namespace std;
#define V 7
int shortest_path(int dist[], int n)
{
 cout<<"Vertex "<<"\t"<<"Distance from Source\n";
 for (int i = 0; i < V; i++)
 cout<<" \t\t \n"<< i<<" \t\t "<<dist[i];
}
int minDist(int dist[], bool Set[])
{
 int min = INT_MAX, min_index;
 for (int i = 0; i < V; i++)
 if (Set[i] == false && dist[i] <= min)
 min = dist[i], min_index = i;
 return min_index;
}
void Dijkstra(int g[V][V], int src)
{
 int dist[V];
 bool Set[V];
 for (int i = 0; i < V; i++)
 dist[i] = INT_MAX, Set[i] = false;
 dist[src] = 0;
 for (int c = 0; c < V- 1; c++)
 {
 int u = minDist(dist, Set);
 Set[u] = true;
 for (int j = 0; j < V; j++)
 if (!Set[j] && g[u][j] && dist[u] != INT_MAX && dist[u]+ g[u][j] < dist[j])
 {
 dist[j] = dist[u] + g[u][j];
 }
 }
 shortest_path(dist, V);
}
int main()
{
- ios_base::sync_with_stdio(false);
 cin.tie(NULL);
 int G[V][V] = {
 { 0, 2, 0, 6, 0, 0, 0},
 { 2, 0, 5, 0, 0, 0, 0},
 { 0, 5, 0, 8, 15, 10, 0},
 { 6, 0, 8, 0, 0, 0, 0, },
{ 0, 0, 15, 9, 0, 6, 6,},
{ 0, 0, 10, 0, 6, 0, 2,},
{ 0, 0, 0, 0, 6, 2, 0,}
};
Dijkstra(G, 0);
return 0;
}
```

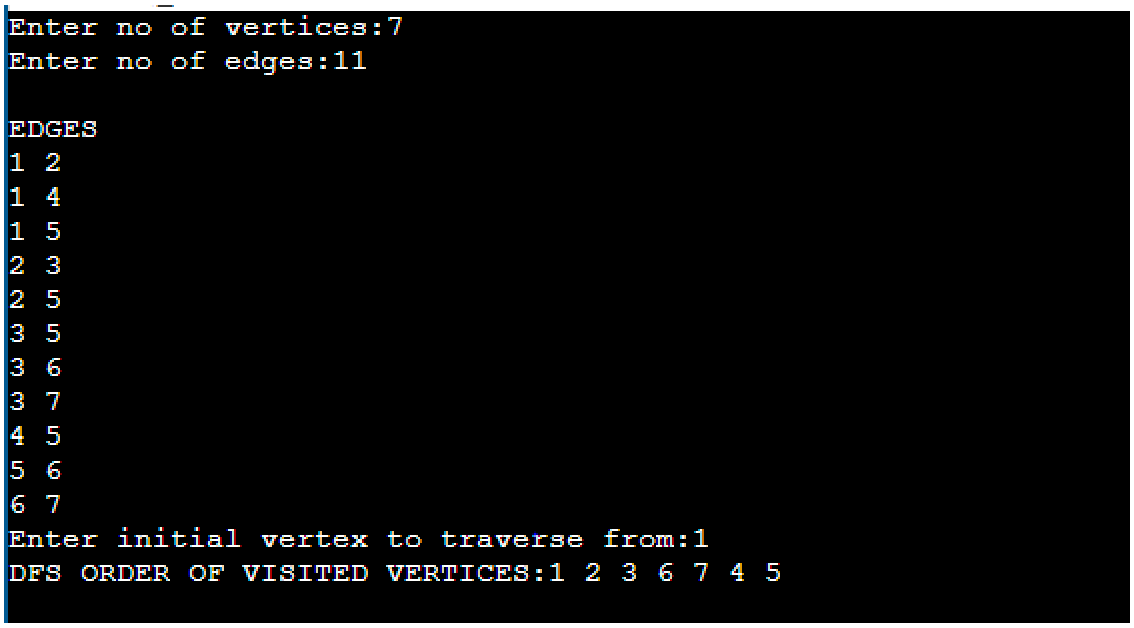
OUTPUTS : -

Bfs:-



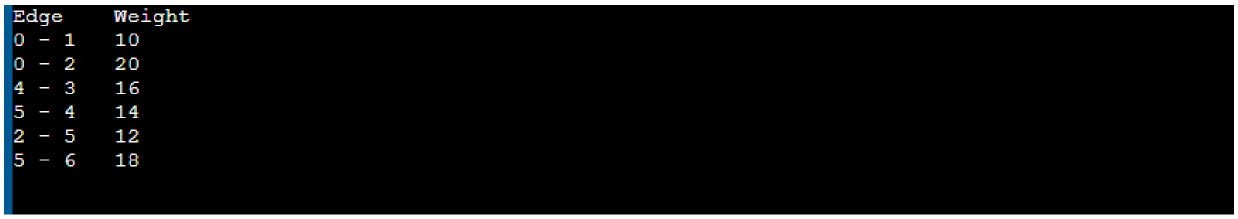
```
Enter no of vertices:7
Enter no of edges:11

EDGES
1 2
1 4
1 5
2 3
2 5
3 5
3 6
3 7
4 5
5 6
6 7
Enter initial vertex to traverse from:1
Visitied vertices:1 2 4 5 3 6 7
```

Dfs:-



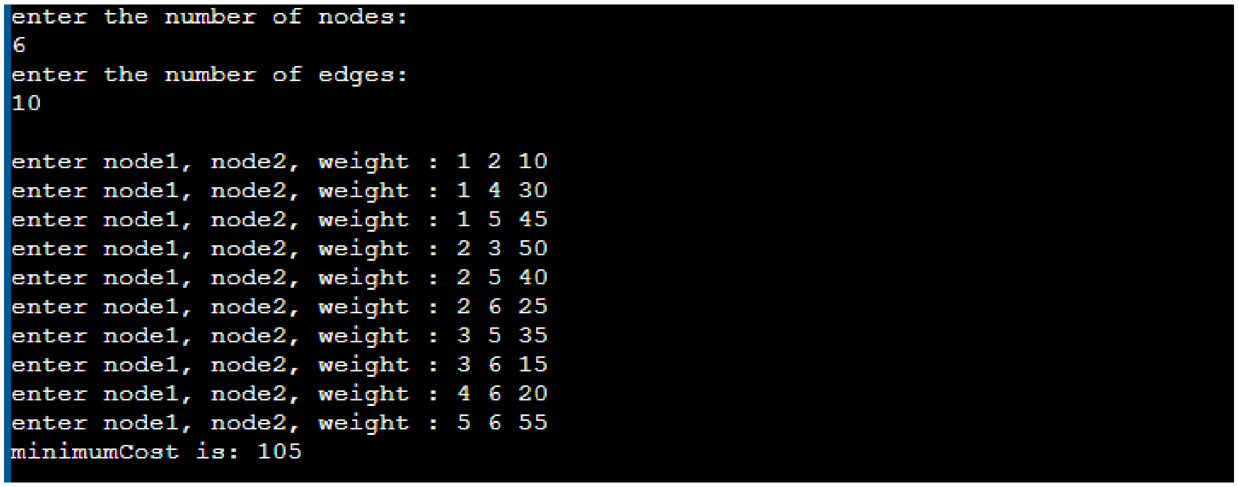
```
Enter no of vertices:7
Enter no of edges:11

EDGES
1 2
1 4
1 5
2 3
2 5
3 5
3 6
3 7
4 5
5 6
6 7
Enter initial vertex to traverse from:1
DFS ORDER OF VISITED VERTICES:1 2 3 6 7 4 5
```

PRIMS:-



```
Edge    Weight
0 - 1   10
0 - 2   20
4 - 3   16
5 - 4   14
2 - 5   12
5 - 6   18
```

Krushkals :



```
enter the number of nodes:
6
enter the number of edges:
10

enter node1, node2, weight : 1 2 10
enter node1, node2, weight : 1 4 30
enter node1, node2, weight : 1 5 45
enter node1, node2, weight : 2 3 50
enter node1, node2, weight : 2 5 40
enter node1, node2, weight : 2 6 25
enter node1, node2, weight : 3 5 35
enter node1, node2, weight : 3 6 15
enter node1, node2, weight : 4 6 20
enter node1, node2, weight : 5 6 55
minimumCost is: 105
```