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 <cstring>
#include <iostream>
using namespace std;

#define INF 9999999

#define V 5

// y adj matrix h
int G[V][V] = {
  {0, 9, 75, 0, 0},
  {9, 0, 95, 19, 42},
  {75, 95, 0, 51, 66},
  {0, 19, 51, 0, 31},
  {0, 42, 66, 31, 0}};

int main() {
  int no_edge;


  int selected[V];
  memset(selected, false, sizeof(selected));

  no_edge = 0;

  selected[0] = true;
```

```
int x;
int y;

cout << "Edge"
  << " : "
  << "Weight";
cout << endl;
while (no_edge < V - 1)

 int min = INF;
 x = 0;
 y = 0;

 for (int i = 0; i < V; i++) {
  if (selected[i]) {
   for (int j = 0; j < V; j++) {
    if (!selected[j] && G[i][j]) {
     if (min > G[i][j]) {
      min = G[i][j];
      x = i;
      y = j;
     }
    }
   }
  }
 }
 cout << x << " - " << y << " :  " << G[x][y];
 cout << endl;
```

```
    selected[y] = true;
    no_edge++;
  }

  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 9



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, 4, 0, 0, 0, 0, 0, 8, 0 },
     { 4, 0, 8, 0, 0, 0, 0, 11, 0 },
     { 0, 8, 0, 7, 0, 4, 0, 0, 2 },
     { 0, 0, 7, 0, 9, 14, 0, 0, 0 },
```

```
    { 0, 0, 0, 9, 0, 10, 0, 0, 0 },
    { 0, 0, 4, 14, 10, 0, 2, 0, 0 },
    { 0, 0, 0, 0, 0, 2, 0, 1, 6 },
    { 8, 11, 0, 0, 0, 0, 1, 0, 7 },
    { 0, 0, 2, 0, 0, 0, 6, 7, 0 }};
    Dijkstra(G, 0);
    return 0;
}
```