**Experiment 6 (Graph Algorithms)**

**Program : Find Minimum Cost Spanning Tree of a given undirected graph using Kruskal’s algorithm.**

#include <cstdio>

#include <vector>

#include <algorithm>

using namespace std;

#define edge pair< int, int >

#define MAX 1001

vector< pair< int, edge > > GRAPH, MST;

int parent[MAX], total, N, E;

int findset(int x, int \*parent)

{

if(x != parent[x])

parent[x] = findset(parent[x], parent);

return parent[x];

}

void kruskal()

{

int i, pu, pv;

sort(GRAPH.begin(), GRAPH.end());

for(i=total=0; i<E; i++)

{

pu = findset(GRAPH[i].second.first, parent);

pv = findset(GRAPH[i].second.second, parent);

if(pu != pv)

{

MST.push\_back(GRAPH[i]);

total += GRAPH[i].first;

parent[pu] = parent[pv];

}

}

}

void reset()

{

for(int i=1; i<=N; i++) parent[i] = i;

}

void print()

{

int i, sz;

sz = MST.size();

for(i=0; i<sz; i++)

{

printf("( %d", MST[i].second.first);

printf(", %d )", MST[i].second.second);

printf(": %d\n", MST[i].first);

}

printf("Minimum cost: %d\n", total);

}

int main()

{

int i, u, v, w;

scanf("%d %d", &N, &E);

reset();

for(i=0; i<E; i++)

{

scanf("%d %d %d", &u, &v, &w);

GRAPH.push\_back(pair< int, edge >(w, edge(u, v)));

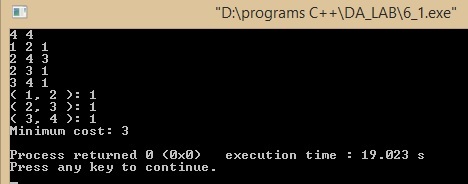
}

kruskal();

print();

return 0;

}



**Program : Find Minimum Cost Spanning Tree of a given undirected graph using Prim’s algorithm.**

#include<bits/stdc++.h>

Using namespace std;

#define MAX 1e8//very large number

typedef vector<int> vi;

typedef pair<int,int> ii;

typedef vector<ii> vii;

typedef vector<vii> vvii;

int main()

{int V;

cin>>V;//Number of vertices

vvii G(V);//The weighted undirected graph

vector<bool>disc(V,false);//whether the vertex is discovered or not

vi d(V,MAX);//all d[] initialised

vi par(V,-1);//parent of the vertex

vii edge;//The edges of the MST

int E;

cout<<"Edges\n";

cin>>E;

cout<<"Input every edge\n";

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

{int u,v,wt;

cin>>u;

cin>>v;

cin>>wt;

u--;

v--;

G[u].push\_back(make\_pair(v,wt));

G[v].push\_back(make\_pair(u,wt));

}

cout<<"Choose start vertex from 1 to "<<V<<endl;

int s;

cin>>s;

s--;

set<ii> Q;

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

Q.insert(make\_pair(d[i],i));

Q.erase(Q.find(make\_pair(d[s],s)));

d[s]=0;

Q.insert(make\_pair(d[s],s));

while(!Q.empty())

{

ii top=\*Q.begin();

Q.erase(Q.begin());

int v=top.second;

disc[v]=true;

if(v!=s)

edge.push\_back(make\_pair(par[v],v));

for(int i=0;i<G[v].size();i++)

{

if(disc[G[v][i].first]==false)

{

int v2=G[v][i].first;

int cost=G[v][i].second;

if(d[v2]>cost)

{

Q.erase(Q.find(make\_pair(d[v2],v2)));

d[v2]=cost;

Q.insert(make\_pair(d[v2],v2));

par[v2]=v;

}

}

}

}

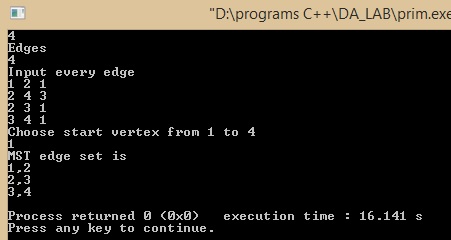
cout<<"MST edge set is"<<endl;

for(int i=0;i<edge.size();i++)

cout<<edge[i].first+ 1<<","<<edge[i].second +1<<endl;

return 0;

}



**Experiment 8 (All Pair’s Shortest Paths)**

**Program : Implement all pair‟s Shortest Paths Problem using Floyd’s-Warshall algorithm.**

#include<bits/stdc++.h>

using namespace std;

#define MAX 2000000

long dist[10010][10010];

int v;

void init()

{

long i,j;

for(i=0;i<v;i++)

{

for(j=0;j<v;j++)

{

if(i==j)

{

dist[i][j]=0;

}

else

{

dist[i][j]=MAX;

}

}

}

}

void floyd\_warshall()

{

long i,j,k;

for(k=0;k<v;k++)

{

for(i=0;i<v;i++)

{

for(j=0;j<v;j++)

{

if(dist[i][j]>dist[i][k]+dist[k][j])

{

dist[i][j]=dist[i][k]+dist[k][j];

}

}

}

}

cout<<"Floyd Warshall distance matrix :"<<endl;

for(i=0;i<v;i++)

{

for(j=0;j<v;j++)

cout<<dist[i][j]<<" ";

cout<<endl;

}

}

int main()

{

long i,j,k,start,last,w;

int edges;

scanf("%ld",&v);

scanf("%ld",&edges);

init();

for(j=0;j<edges;j++)

{

scanf("%ld",&start);

scanf("%ld",&last);

scanf("%ld",&w);

dist[start-1][last-1]=w;

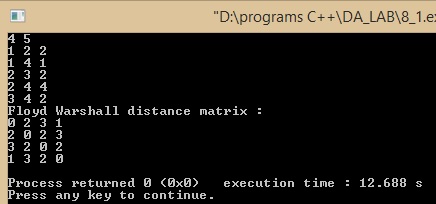
dist[last-1][start-1]=w;

}

floyd\_warshall();

return 0;

}



**Experiment 9 (BFS and DFS problems)**

**Program : Write an algorithm to print all the nodes reachable from a given starting node in a digraph using BFS method.**

#include<bits/stdc++.h>

using namespace std;

int n;

int e;

int graph[1010][1010];

int visited[1010];

void bfs(int x)

{

int i,j,k;

queue<int> q;

q.push(x);

visited[x]=1;

while(!q.empty())

{

x=q.front();

q.pop();

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

{

if(graph[x][i]==1&&!visited[i])

{

visited[i]=1;

cout<<i+1<<" ";

q.push(i);

}

}

}

}

int main()

{

int i,j,k;

int x,y;

cin>>n>>e;

for(i=0;i<e;i++)

{

cin>>x>>y;

x--;

y--;

graph[x][y]=1;

}

cout<<"Enter the node : "<<endl;

cin>>x;

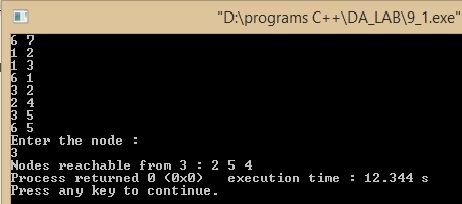
cout<<"Nodes reachable from "<<x<<" : ";

x--;

bfs(x);

return 0;

}



**Program : Obtain the Topological ordering of vertices in a given digraph.**

#include<bits/stdc++.h>

using namespace std;

int n;

int e;

int graph[1010][1010];

int visited[1010];

stack<int> s;

void dfs(int x)

{

int i;

visited[x]=1;

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

{

if(graph[x][i]&&!visited[i])

dfs(i);

}

s.push(x);

}

int main()

{

int i,j,k;

int x,y;

cin>>n>>e;

for(i=0;i<e;i++)

{

cin>>x>>y;

x--;

y--;

graph[x][y]=1;

}

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

if(!visited[i])

dfs(i);

cout<<"Elements in topological sorted order :"<<endl;

while(!s.empty())

{

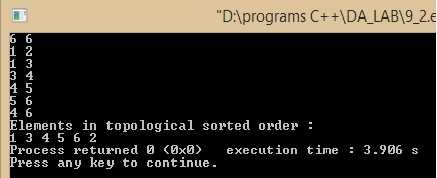
cout<<s.top()+1<<" ";

s.pop();

}

return 0;

}



**Program : Check whether a given graph is connected or not using DFS method.**

#include<bits/stdc++.h>

using namespace std;

int n;

int e;

int graph[1010][1010];

int visited[1010];

void dfs(int x)

{

int i;

visited[x]=1;

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

{

if(graph[x][i]&&!visited[i])

dfs(i);

}

}

int main()

{

int i,j,k;

int x,y;

cin>>n>>e;

for(i=0;i<e;i++)

{

cin>>x>>y;

x--;

y--;

graph[x][y]=1;

graph[y][x]=1;

}

dfs(0);

int ans=0;

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

if(!visited[i])

ans=1;

if(ans==0)

{

cout<<"Connected graph"<<endl;

}

else

cout<<"Not connected"<<endl;

return 0;

}

