**DFS ADJACENCY CODE**

#include<bits/stdc++.h>

using namespace std;

void DFS(int start , vector<bool>&visited , vector<int> \*adj)

{ if(visited[start])

return;

visited[start] = true;

cout << start << " ";

for (auto & i : adj[start])

if (!visited[i])

DFS(i, visited , adj);

return;

}

int main()

{

int v , e;

cin>>v>>e;

vector<int> adj[v];

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

{

int f,s;

cin>>f>>s;

adj[f].push\_back(s);

adj[s].push\_back(f);

}

vector<bool> visited(v , false);

DFS( 0 , visited , adj);

return 0;

}

**BFS ADJACENCY CODE**

void BFS(int start , vector<int> \* adj , vector<int> & visited)

{

queue<int> queue;

queue.push(start);

while(!queue.empty())

{

s = queue.front();

cout << s << " ";

queue.pop();

for (auto & it : adj[start])

{

if (!visited[i])

{

visited[i] = true;

queue.push(i);

}

}

}

}

**HAS PATH ADJACENCY CODE**

#include <bits/stdc++.h>

using namespace std;

bool haspath(int start , int end , vector<bool> & visited , vector<int> \* adj)

{

if(start == end)

return true;

visited[start] = true;

for(auto & it : adj[start])

{

if(it == end)

return true;

}

for(auto & it : adj[start])

{

if(!visited[it])

{ bool ans = haspath(it , end , visited , adj);

if(ans)

return true;

}

}

return false;

}

int main() {

int V, E;

cin >> V >> E;

vector<int> adj[V];

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

{

int f , s;

cin>>f>>s;

adj[f].push\_back(s);

adj[s].push\_back(f);

}

int start , end;

cin>>start>>end;

vector<bool> visited(V,0);

if(haspath(start , end ,visited , adj ))

cout<<"true";

else

cout<<"false";

return 0;

}

**GETPATH DFS ADJACENCY LIST**

#include <bits/stdc++.h>

using namespace std;

vector<int> fn(vector<int> \* adj , int v1 , int v2, vector<int> & visited)

{ if(visited[v1])

{

return {};

}

if(v1==v2)

{

vector<int> ans;

ans.push\_back(v1);

return ans;

}

visited[v1] = 1;

for(auto & it : adj[v1])

{

if(!visited[it])

{

vector<int> smallans = fn(adj , it , v2 , visited );

if(smallans.size() != 0)

{

smallans.push\_back(v1);

return smallans;

}

}

}

return {};

}

int main() {

int v , e;

cin>>v>>e;

vector<int> adj[v];

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

{

int a , b;

cin>>a>>b;

adj[a].push\_back(b);

adj[b].push\_back(a);

}

int v1 , v2;

cin>>v1>>v2;

vector<int> visited(v ,0);

vector<int> anss = fn(adj ,v1,v2 , visited);

if(anss.size() == 0)

{

cout<<" ";

return 0;

}

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

{

cout<<anss[i]<<" ";

}

return 0;

}

**GETPATH BFS**

#include <bits/stdc++.h>

using namespace std;

void bfs(vector<int>\* adj , int x , int y , vector<int> & visited , int v)

{

queue<int> q;

q.push(x);

visited[x] = 1;

bool getpath = false;

vector<int> parent(v , -1);

while(!q.empty() && !getpath)

{

int front = q.front();

q.pop();

for(auto & it : adj[front])

{if(!visited[it])

{ parent[it] = front;

q.push(it);

visited[it] = 1;

if(it == y)

{

getpath = true;

break;

}

}

}

}

if(getpath = false)

return;

vector<int> output;

int current = y;

output.push\_back(y);

while(current != x)

{

current = parent[current];

output.push\_back(current);

}

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

{

cout<<output[i]<<" ";

}

return;

}

int main()

{

int v, e, tempX, tempY;

cin >> v >> e;

vector<int> adj[v];

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

{

int a , b;

cin>>a>>b;

adj[a].push\_back(b);

adj[b].push\_back(a);

}

cin>>tempX>>tempY;

vector<int> visited(v ,0);

bfs(adj , tempX , tempY , visited , v);

return 0;

}

**CHECK IF ALL CONNECTED ADJACENCY MATRIX**

#include<bits/stdc++.h>

using namespace std;

void fn(vector<int> \* adj, vector<int>& visited , int start)

{

visited[start] = 1;

for(auto & it : adj[start])

{

if(!visited[it])

{

visited[it] = 1;

fn(adj , visited , it);

}

}

return;

}

int main()

{

int v,e,i,j;

cin>>v>>e;

vector<int> adj[v];

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

{

int f,s;

cin>>f>>s;

adj[f].push\_back(s);

adj[s].push\_back(f);

}

vector<int> visited(v ,0);

fn(adj , visited , 0);

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

{

if(visited[i]==0)

{

cout<<"false";

return 0;

}

}

cout<<"true";

return 0;

}

**PRINT ALL CONNECTED COMPONENT**

#include<bits/stdc++.h>

using namespace std;

void fn(vector<int> \* adj, vector<int>& visited , int start)

{

cout<<start<<" ";

visited[start] = 1;

for(auto & it : adj[start])

{

if(!visited[it])

{

visited[it] = 1;

fn(adj , visited , it);

}

}

cout<<endl;

return;

}

int main()

{

int v,e,i,j;

cin>>v>>e;

vector<int> adj[v];

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

{

int f,s;

cin>>f>>s;

adj[f].push\_back(s);

adj[s].push\_back(f);

}

vector<int> visited(v ,0);

fn(adj , visited , 0);

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

{

if(visited[i]==0)

{

fn(adj , visited , i);

}

}

return 0;

}

**DIKSHTRA ALGORITHM**

#include<bits/stdc++.h>

using namespace std;

int findminvertex(vector<int> distance , vector<int> visited , int n)

{ int minvertex = -1;

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

{

if(!visited[i] && (minvertex == -1 || distance[i]<distance[minvertex]))

{

minvertex = i;

}

}

return minvertex;

}

void dikshtra(vector<vector<int>> edges , int n)

{

vector<int> distance(n , INT\_MAX);

vector<int> visited(n , 0);

distance[0] = 0;

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

{

int minvertex = findminvertex(distance , visited , n);

visited[minvertex] = 1;

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

{

if(edges[minvertex][j] && visited[j]==0)

{

int dist = distance[minvertex]+edges[minvertex][j];

if(dist<distance[j])

{

distance[j] = dist;

}

}

}

}

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

{

cout<<i<<" "<<distance[i]<<endl;

}

}

int main()

{

int n , e;

cin>>n>>e;

vector<vector<int>> edges(n , vector<int>(n, 0));

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

{

int f , s , w;

cin>>f>>s>>w;

edges[f][s] = w;

edges[s][f] =w ;

}

dikshtra(edges , n);

return 0;

}