**Graph Resprentation**

**#include <iostream>**

**using namespace std;**

**bool A[10][10];**

**void initialize()**

**{**

**for(int i = 0; i < 10; ++i)**

**for(int j = 0; j < 10; ++j)**

**A[i][j] = false;**

**}**

**int main()**

**{**

**int x, y, nodes, edges;**

**initialize(); //Since there is no edge initially**

**cin >> nodes; //Number of nodes**

**cin >> edges; //Number of edges**

**for(int i = 0; i < edges; ++i)**

**{**

**cin >> x >> y;**

**A[x][y] = true; //Mark the edges from vertex x to vertex y**

**}**

**if(A[3][4] == true)**

**cout <<"There is an edge between 3 and 4" << endl;**

**else**

**cout << "There is no edge between 3 and 4" << endl;**

**return 0;**

**}**

**OutputThere is an edge between 3 and 4.**

**Graph Resprentation**

**#include<bits/stdc++.h>**

**using namespace std;**

**vector <int> adj[10];**

**int main()**

**{**

**int x, y, nodes, edges;**

**cin >> nodes;**

**//Number of nodes**

**cin >> edges;**

**//Number of edges**

**for(int i = 0; i < edges; ++i)**

**{**

**cin >> x >> y;**

**adj[x].push\_back(y);**

**//Insert y in adjacency list of x**

**}**

**for(int i = 1; i <= nodes; ++i)**

**{**

**cout << "Adjacency list of node " << i << ": ";**

**for(int j = 0; j < adj[i].size(); ++j)**

**{**

**if(j == adj[i].size() - 1)**

**cout << adj[i][j] << endl;**

**else**

**cout << adj[i][j] << " --> ";**

**}**

**}return 0; }**

**BFS(Breadth-First-Search)**

**#include<bits/stdc++.h>**

**using namespace std;**

**vector<int>adj[10007];**

**int visited[10007];**

**int lavel[10007];**

**void bfs(int source, int destination)**

**{**

**queue<int>Q;**

**Q.push(source);**

**while(!Q.empty())**

**{**

**int node=Q.front();**

**Q.pop();**

**visited[node]=1;**

**cout<<node<<" ";**

**if(node==destination)**

**{**

**cout<<"Minimum path= "<<lavel[node]<<endl;**

**return;**

**}**

**for(int i=0; i<adj[node].size(); i++)**

**{**

**if(visited[adj[node][i]]==0)**

**{**

**visited[adj[node][i]]=1;**

**Q.push(adj[node][i]);**

**lavel[adj[node][i]]=lavel[node]+1;**

**}**

**}**

**}**

**cout<<"Node is not available!! '\_' "<<endl;**

**}**

**int main()**

**{**

**int vertex,edge;**

**cin>>vertex>>edge;**

**for(int i=0;i<edge;i++)**

**{**

**int a,b;**

**cin>>a>>b;**

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

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

**}**

**int source,destination;**

**cin>>source>>destination;**

**bfs(source,destination);}**

**DFS(Depth-First-Search)**

**#include<bits/stdc++.h>**

**using namespace std;**

**vector<int>adj[100];**

**int visited[100];**

**int c=0;**

**void dfs(int n)**

**{**

**visited[n]=1;**

**cout<<n<<endl;//visited node**

**for(int i=0; i<adj[n].size(); i++)**

**{**

**if(visited[adj[n][i]]==0)**

**dfs(adj[n][i]);**

**}**

**return;**

**}**

**int main()**

**{**

**int edge,node;**

**cin>>node>>edge;**

**int i;**

**for(i=0; i<edge; i++)**

**{**

**int a,b;**

**cin>>a>>b;**

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

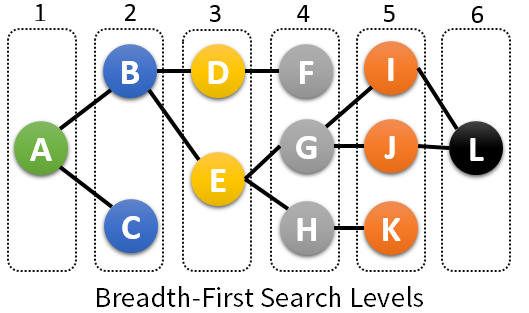
**adj[b].push\_back(a);**

**}**

**dfs(1);//starting point**

**}**

****



**How many Components Here!!!!**

**#include<bits/stdc++.h>**

**using namespace std;**

**vector<int>adj[1000];**

**int visited[1000];**

**int mark[1000];**

**void dfs(int s)**

**{**

**visited[s]=1;**

**for(int i=0; i<adj[s].size(); i++)**

**{**

**if(visited[adj[s][i]]==0)**

**dfs(adj[s][i]);**

**}**

**return;**

**}**

**int main()**

**{**

**int edge;**

**cin>>edge;**

**int i;**

**vector<int>v1;**

**memset(mark,-1,1000);**

**for(i=0; i<edge; i++)**

**{**

**int a,b;**

**cin>>a>>b;**

**if(mark[a]==-1)**

**{**

**mark[a]=1;**

**v1.push\_back(a);**

**}**

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

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

**}**

**int c=0;**

**for(i=0; i<v1.size(); i++)**

**{**

**if(visited[v1[i]]==0)//checking this node is visited or not,**

**{**

**dfs(v1[i]);//if not visited then call dfs**

**c++;//new component arise**

**}**

**}**

**cout<<c<<endl;**

**}**

**<<c<<endl;**

**}**

**DisJoint Set Unon(DSU)-Finding Representative of a componenet**

**#include<bits/stdc++.h>**

**using namespace std;**

**typedef long long ll;**

**#define an 100007**

**ll parent[an];**

**void makeset(ll n)//nejeder parent nejei**

**{**

**ll i;**

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

**parent[i]=i;**

**}**

**ll FindRepresentative(ll r)**

**{**

**if(parent[r]==r)**

**{**

**return r;**

**}**

**parent[r]=FindRepresentative(parent[r]);**

**return parent[r];**

**}**

**void unioN(ll a,ll b)**

**{**

**ll u,v;**

**u=FindRepresentative(a);**

**v=FindRepresentative(b);**

**if(u!=v)**

**{**

**parent[u]=v;**

**}**

**}**

**int main()**

**{**

**ll n,m,i;**

**cin>>n>>m;**

**makeset(n);**

**for(i=0; i<m; i++)**

**{**

**int x,y;**

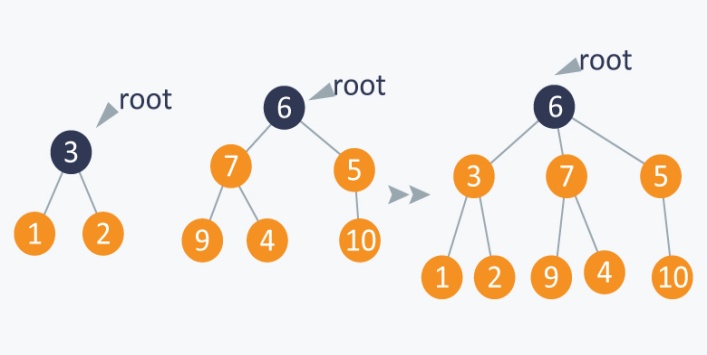
**cin>>x>>y;**

**unioN(x,y);//x and y Wants to be friend**

**}**

**return 0;**

**}**

****

**Bipartie**

#include<bits/stdc++.h>

using namespace std;

typedef long long ll;

ll color[100007];

vector<ll>adj[100007];

ll l,r;

void bipa(ll x)

{

color[x]=1;

l++;

queue<ll>q;

q.push(x);

while(!q.empty())

{

ll u=q.front();

q.pop();

for(ll i=0; i<adj[u].size(); i++)

{

ll v=adj[u][i];

if(color[v]==-1)

{

if(color[u]==1)

{

color[v]=2;

r++;

}

else

{

color[v]=1;

l++;

}

q.push(v);

}

}

}

}

int main()

{

ll n;

cin>>n;

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

{

ll a,b;

cin>>a>>b;

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

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

}

memset(color,-1,sizeof color);

bipa(1);

//l means left group and r means right group

}

**Dijkastra(Priority\_queue)**

**void dijkastra(int start,int m)**

**{**

**priority\_queue<pair<int,int>, vector<pair<int,int>>,**

**greater<pair<int,int>>>q;**

**q.push(make\_pair(0,start));**

**dis[start]=0;**

**while(!q.empty())**

**{**

**int current=q.top().second;**

**/\***

**if(current==m)**

**{**

**cout<<dis[m]<<endl; //for a perticular node**

**return;**

**}\*/**

**q.pop();**

**for(int i=0; i<adj[current].size(); i++)**

**{**

**int u=adj[current][i];**

**if(dis[current]+edge[current][u]<dis[u])**

**{**

**dis[u]=dis[current]+edge[current][u];**

**q.push(make\_pair(dis[u],u));**

**}**

**}**

**}**

**}**

**Prim’s Algorithm:**

**vector<pii>v[10000];**

**Tropological Sort**

**#include <bits/stdc++.h>**

**using namespace std;**

**void addedge(list<int>\* ls, int x, int y)**

**{**

**ls[x].push\_back(y);**

**}**

**string topological\_sort(list<int>\* ls, int k)**

**{**

**char arr[k];**

**stack<int> s;**

**set<int> st;**

**int ind = k - 1;**

**for (int i = k - 1; i >= 0; i--) {**

**if (st.find(i) == st.end()) {**

**s.push(i);**

**st.insert(i);**

**//check all the non visited nodes**

**while (!s.empty()) {**

**int p = s.top();**

**list<int>::iterator it;**

**int temp = 0;**

**//check its adjacent non visited nodes**

**for (it = ls[p].begin(); it != ls[p].end(); it++) {**

**if (st.find(\*it) == st.end()) {**

**st.insert(\*it);**

**s.push(\*it);**

**temp = 1;**

**}**

**}**

**//if all adjaceny nodes are visited then pop that element from stack**

**if (temp == 0) {**

**char ch = (char)(p + 'A');**

**arr[ind] = ch;**

**ind--;**

**s.pop();**

**}**

**}**

**}**

**}**

**return arr;**

**}**

**int main()**

**{**

**int k = 7;**

**list<int> ls[k];**

**addedge(ls, 0, 2);**

**addedge(ls, 0, 3);**

**addedge(ls, 1, 2);**

**addedge(ls, 1, 4);**

**addedge(ls, 4, 5);**

**addedge(ls, 3, 5);**

**addedge(ls, 5, 6);**

**cout << topological\_sort(ls, 7) << endl;**

**return 0;**

**}**