EASY :

1. Implement BFS and DFS.

```
#include<bits/stdc++.h>
using namespace std;
vector<int>adj[10000007];
void addEdge(int v, int w)
{
    adj[v].push_back(w);
}
void BFS(int st,int n)
{
    vector<bool>vis(n+1,0);
    for(int i=0;i<=n;i++)
    vis[i]=false;
    vis[st]=true;
    queue<int>q;
    q.push(st);
    while(!q.empty())
    {
       int curr=q.front();
        cout<<curr<<" ";
        q.pop();
        for(auto it:adj[curr])
        if(!vis[it])
            {
                vis[it] = true;
                q.push(it);
            }
    }
}
unordered_map<int,bool>visited;
void DFS(int v)
{
    visited[v] = true;
    cout<<v<<" ";
    for (auto it:adj[v])
        if (!visited[it])
            DFS(it);
}
int main()
{
    addEdge(0, 1);
    addEdge(0, 2);
    addEdge(1, 2);
    addEdge(2, 0);
    addEdge(2, 3);
    addEdge(3, 3);
    cout<<"BFS is : "; BFS(2,4);
    cout<<endl<<"DFS is : "; DFS(2);
    return 0;
}
```

2.   Path in directed graph

```
void solve1(int i,int a,vector<int>adj[],vector<int>&vis)
{
    vis[i]=1;
    for(auto it:adj[i])
    {
        if(it!=i && vis[it]!=1)
        solve1(it,a,adj,vis);
    }return;
}
int Solution::solve(int A, vector<vector<int> > &B)
{
    vector<int>adj[200007];
    vector<int>vis(200007);
    int n=B.size();
    for(int i=0;i<n;i++)
    adj[B[i][0]].push_back(B[i][1]);

    for(int i=0;i<200007;i++)
    vis[i]=0;
    solve1(1,A,adj,vis);
    return vis[A]==1;
}
```

MEDIUM :

1. Find circle in directed graph/ topological sort

A. BFS solution : // Khan’s algorithm (topological sort)

```
    class Solution {
public:
    vector<int> findOrder(int numCourses, vector<vector<int>>&
prerequisites)
    {
        int n=numCourses;
        vector<int>adj[n];
        vector<int>degree(n,0);
         vector<int>ans;
        queue<int>q;


        for(int i =0; i< prerequisites.size(); i++ )
        adj[prerequisites[i][1]].push_back(prerequisites[i][0]);

        for(int i =0; i < prerequisites.size(); i++)
        degree[prerequisites[i][0]]++;

         for(int i=0;i<degree.size();i++)
        if(degree[i] == 0)
        q.push(i);
        while(!q.empty())
        {
           int temp=q.front();
            q.pop();
            for(auto it : adj[temp])
            {
                 degree[it]--;
                 if(degree[it] == 0)
                 q.push(it);
            }
            ans.push_back(temp);
        }
        if (ans.size()!=n)
        ans.clear();
        return ans;
    }
};
```

B. DFS Solution :

```
class Solution{
    public:
    bool cyc_detect(unordered_map<int,vector<int>>& adj, int u, 
vector<int>& vis, stack<int>& st)
    {
        vis[u]=1;
        for(auto v:adj[u])
        {
            if(vis[v]==1 || (vis[v]==0 && cyc_detect(adj,v,vis,st)))
            return true;
        }
        vis[u]=2;
        st.push(u);
        return false;
    }
    vector<int> findOrder(int numCourses, vector<vector<int>>& 
prerequisites)
    {
        unordered_map<int,vector<int>>adj;
        vector<int>ans;

        for(int i=0;i<prerequisites.size();i++)
        adj[prerequisites[i][1]].push_back(prerequisites[i][0]);

        vector<int>vis(numCourses,0);
        stack<int>st;

        for(int i=0;i<numCourses;i++)
        {
            if(vis[i]==0)
            {
                if(cyc_detect(adj,i,vis,st))
                return {};
            }
        }
        while(!st.empty())
        {
            ans.push_back(st.top());
            st.pop();
        }
        return ans;
    }
};
```

2. Find circle in undirected graph.

A. DFS solution:

```
class Solution {
public:
    vector<int>parent;
    int find_parent(int u)
    {
         if(parent[u] == u)
         return u;
         return parent[u] = find_parent(parent[u]);
    }
    vector<int> findRedundantConnection(vector<vector<int>>& edges)
    {
        int n=edges.size();
        for(int i=0;i<=n;i++)
        parent.push_back(i);

        for(auto it : edges)
        {
            int p1=find_parent(it[0]);
            int p2=find_parent(it[1]);
            if(p1!=p2)
            parent[p1]=p2;
            else
            return it;
        }
        return {};
    }
};
```

3. Search in a maze.

```
class Solution{
    public:
    vector<string>ans;
     void solve(vector<vector<int>> &m, int n,string s,int r,int
c,vector<vector<bool>> &vis{
        if(r==n-1 && c==n-1)
        {
            ans.push_back(s);
            return ;
        }

        if(r+1<n && !vis[r+1][c])
        {
            if(m[r+1][c]==1)
            {   vis[r+1][c]=1;
                solve(m,n, s+"D",r+1,c,vis);
                vis[r+1][c]=0;
           }
        }

        if(c-1>=0 && !vis[r][c-1])
        {
            if(m[r][c-1]==1)
            {
                vis[r][c-1]=1;
                solve(m,n,s+"L",r,c-1,vis);
                vis[r][c-1]=0;
            }
        }

        if(c+1<n && !vis[r][c+1])
        {
            if(m[r][c+1]==1)
            {
                vis[r][c+1]=1;
                solve(m,n,s+"R",r,c+1,vis);
                vis[r][c+1]=0;
            }
        }

        if(r-1>=0 && !vis[r-1][c])
    {
      if(m[r-1][c]==1)
      {
        vis[r-1][c]=1;
        solve(m,n,s+"U",r-1,c,vis);
        vis[r-1][c]=0;
      }
    }
  }
  vector<string> findPath(vector<vector<int>> &m, int n)
  {
    vector<vector<bool>>vis (n, vector<bool> (n,0));
    if(m[n-1][n-1]==0||m[0][0]==0)
    return ans;
    vis[0][0]=1;
    string s="";
    solve(m,n,s,0,0,vis);
    return ans;
  }
};
```

4. Flood fill Algo

```
class Solution {
public:
void solve(vector<vector<int>>& image,int x,int y,int r,int c,int
nclr,int clr)
  {

    if(x<0||y<0||x>r||y>c||image[x][y]!=clr)
    return;

    image[x][y]=nclr;

    solve(image,x-1,y,r,c,nclr,clr);
    solve(image,x+1,y,r,c,nclr,clr);
    solve(image,x,y-1,r,c,nclr,clr);
    solve(image,x,y+1,r,c,nclr,clr);
  }

  vector<vector<int>> floodFill(vector<vector<int>>& image, int sr,
int sc, int newColor) {

    int clr=image[sr][sc];
```

```
        int r=image.size();
        int c=image[0].size();

        if(clr==newColor)
        return image;

        solve(image,sr,sc,r-1,c-1,newColor,clr);
        return image;
    }
};
```

5. Clone a graph.

```
class Solution {
public:
    Node* cloneGraph(Node* node)
    {
         queue<Node*> q;
         unordered_map<int, Node*>mp;
         unordered_set<int>vis;
         if(node==0)
         return node;
         Node* start= new Node(node->val);
         q.push(node);
         mp[node->val]=start;
         vis.insert(node->val);

        while(!q.empty())
        {
            Node* cur = q.front();
            Node* begin = mp[cur->val];

            for(auto it : cur->neighbors)
            {
                if(vis.find(it->val)==vis.end())
                {
                    vis.insert(it->val);
                    q.push(it);
                }

                if(mp.find(it->val)==mp.end())
                mp[it->val]= new Node(it->val);
                begin->neighbors.push_back(mp[it->val]);
            }
            q.pop();
        }
        return start;
    }
};
```

6. Making wired connection .

```
class Solution {
public:
    vector<int>adj[100005];
    void solve(int child, vector<int>&vis)
    {
        vis[child]=true;
        for(auto it:adj[child])
        {
            if(!vis[it])
            solve(it,vis);
        }
    }
    int makeConnected(int n, vector<vector<int>>& connections)
    {
         int l=connections.size();
         vector<int>vis(n,0);
         if(l<(n-1))
         return -1;
         else
         {
            for(int i=0;i<l;i++)
            {
                int x=connections[i][0];
                int y=connections[i][1];

                adj[x].push_back(y);
                adj[y].push_back(x);

            }
            int count=0;
            for(int i=0;i<n;i++)
            {
                if(!vis[i])
                {
                    solve(i,vis);
                    count++;
                }
            }
        return count-1;
        }
    }
};
```

7. Find number of islands.

```
class Solution {
  public:
    void solve(vector<vector<char>>& grid, int x, int y,int n,int m)
    {
        grid[x][y] = '0';
        int dx[] = {0, 1, 1, 1, 0, -1, -1, -1};
        int dy[] = {1, 1, 0, -1, -1, -1, 0, 1};

        for(int i=0; i<8; i++)
        {
            int tmp = x+dx[i];
            int tmp1 = y+dy[i];
            if(tmp>=0 && tmp<n && tmp1>=0 && tmp1<m &&
grid[tmp][tmp1]=='1')
            solve(grid, tmp, tmp1,n,m);
        }
    }

    int numIslands(vector<vector<char>>& grid)
    {
        int n=grid.size();
        int m=grid[0].size();
        int ans=0;

        for(int i=0; i<n; i++)
        {
            for(int j=0; j<m; j++)
            {
                if(grid[i][j]=='1')
                {
                    solve(grid,i,j,n,m);
                    ans++;
                }
            }
        }
        return ans;
    }
  };
```

8. Detect negative circle (bellman ford algo)

```
class Solution {
public:
   int isNegativeWeightCycle(int n, vector<vector<int>>edges){
       vector<int>ans(n, INT_MAX);
        ans[0] = 0;

        for(int i=1; i<=n-1; i++)
        {
            for(auto it : edges)
            {

                if(ans[it[0]]==INT_MAX)
                continue;
                if(ans[it[0]] + it[2] < ans[it[1]])
                ans[it[1]] = ans[it[0]] + it[2];
            }
        }
        for(auto& it: edges)
        {
            if(ans[it[0]] + it[2] < ans[it[1]])
            return 1;
        }
        return 0;
   }
};
```

9. Floyd warshall algorithm

```
class Solution {
  public:
   void shortest_distance(vector<vector<int>>&matrix){
       int n = matrix.size();
```

```
        for(int k=0; k<n; k++)
        for(int i=0; i<n; i++)
        {
            for(int j=0; j<n; j++)
            {
            if(matrix[i][k]==-1 || matrix[k][j]==-1)
            continue;
            if(matrix[i][j]==-1)
            matrix[i][j] = matrix[i][k] + matrix[k][j];
            else
            matrix[i][j] = min(matrix[i][j],
            matrix[i][k]+matrix[k][j]);
            }
        }
    }
};
```

10. Snake and ladder problem

```
class Solution {
public:
    int snakesAndLadders(vector<vector<int>>& board)
    {
        unordered_map<int, int>mp;
        int  n = board[0].size();
        int dir = 1;
        int index = 1;
        for(int i=n-1;i>=0;i--)
        {
            if(dir==1)
            {
                for(int j = 0; j < n; j++)
                {
                    if(board[i][j] != -1)
                    mp[index] = board[i][j];
                    index++;
                }
            }
            else
            {
                for(int j = n - 1; j >= 0; j--)
                {
                    if(board[i][j] != -1)
                    mp[index] = board[i][j];
                    index++;
                }
            }
            dir=-dir;
         }
        int m=n*n+1;
        vector<int>dist(m, -1);
        queue<int>q;
        q.push(1);
        dist[1] = 0;
        while (!q.empty())
        {
            int v=q.front();
            q.pop();
            if (v== n*n)
            return dist[n*n];
            for (int k=1;k<=6;k++)
            {
                int temp=v+k;
                if (temp>n*n)
                break;
                if (mp.find(temp) != mp.end())
                temp = mp[temp];
                if (dist[temp] == -1)
                {
                    dist[temp] = dist[v] + 1;
                    q.push(temp);
                }
             }
        }
       return -1;
    }
};
```

11. Journey to the moon

```
long long int find(long long int x)
{
    if(par[x]==x)
    return x;
    return par[x]=find(par[x]);
```

```
}
void unioon(long long int x,long long int y)
{
    long long int lx=find(x);
    long long int ly=find(y);
    if(rannk[lx]<rannk[ly])
    par[lx]=ly;
    else if(rannk[lx]>rannk[ly])
    par[ly]=lx;
    else
    {
        par[lx]=ly;
        rannk[ly]++;
    }
}
long long int journeyToMoon(int n, vector<vector<int>> astronaut)
{
     for(long long int i=0;i<n;i++)
    {
        par[i]=i;
        rannk[i]=i;
    }
    int m=astronaut.size();
    for(long long int i=0;i<m;i++)
    unioon(astronaut[i][0],astronaut[i][1]);
    long long int nn=n;
    long long int ans=((nn*(nn-1))/2);
    unordered_map<long long int,long long int>mp;
    for(long long int i=0;i<n;i++)
    {
        long long int pres=find(i);
        mp[pres]++;
    }
    for(auto it:mp)
    {
        long long int val=it.second;
        ans=ans-(long long int)((val*(val-1))/2);
    }
    return ans;
}
```

12. Oliver and the game

```
   vector<int>s_Time;
   vector<int>e_Time;
   int tim=0;
   void DFS(vector<int> graph[], int start, vector<bool> &vis)
   {
      vis[start] = true;
      s_Time[start] = tim++;

      for(auto node : graph[start])
      if(!vis[node])
      DFS(graph, node, vis);

      e_Time[start] = tim++;
   }
   bool check(int x, int y)
   {
      if(s_Time[x]>s_Time[y] && e_Time[x]<e_Time[y])
      return true;
      return false;
   }
   int main()
   {
      int n;
      cin>>n;
      s_Time.resize(n+1);
      e_Time.resize(n+1);
      vector<int>adj[n+1];

      for(int i=0;i<n-1;i++)
      {
          int x, y;
```

```
       cin>>x>>y;
       adj[x].push_back(y);
   }
   vector<bool>vis(n+1);
   DFS(adj, 1, vis);
   int q;
   cin>>q;
   while(q--)
   {
       int type, x, y;
       cin>>type>>x>>y;
       if(type == 0)
       {
            if(check(y, x))
            cout<<"YES"<<"\n";
            else
            cout<<"NO"<<"\n";
       }
       else
       {
           if(check(x, y))
           cout<<"YES"<<"\n";
           else
           cout<<"NO"<<"\n";
       }
   }
   return 0;
}
```

13. M-coloring problem

```
bool check(int node, int c[], bool g[101][101], int n, int col)
{
    for(int i=0;i<n;i++)
    {
        if(i != node && g[i][node] == 1 && c[i] == col)
        return false;
    }
    return true;
}

bool solve(int node, int c[], bool g[101][101], int m, int v) {
    if(node == v)
    return true;
    for(int i = 1;i<=m;i++)
    {
        if(check(node, c, g, v, i))
        {
            c[node] = i;
            if(solve(node+1, c, g, m, v))
            return true;
            c[node] = 0;
        }
     }
    return false;
}
bool graphColoring(bool graph[101][101], int m, int V)
{
    int c[V] = {0};
    if(solve(0,c,graph,m,V))
    return true;
    return false;
}
```