#include<iostream>

#include<vector>

#include<queue>

#include<omp.h>

using namespace std;

class Graph

{

    private:

    int V;

    vector<vector<int>> adj;

    public:

    Graph (int V): V(V)

    {

        adj.resize(V);

    }

    void addEdge(int u, int v)

    {

        adj[u].push\_back(v);

    }

    void bfs(int start)

    {

        vector<bool> visited(V, false);

        queue<int> q;

        visited[start] = true;

        q.push(start);

        while(!q.empty())

        {

            int u = q.front();

            q.pop();

            cout<<u<<" ";

            #pragma omp parallel for

            for(int v : adj[u])

            {

                if(!visited[v])

                {

                    visited[v] = true;

                    q.push(v);

                }

            }

        }

        cout<<endl;

    }

    void dfsUtil(int v, vector<bool>& visited)

    {

        visited[v] = true;

        cout<<v<<" ";

        #pragma omp parallel for

        for(int u : adj[v])

        {

            if(!visited[u])

            {

                dfsUtil(u, visited);

            }

        }

    }

    void dfs(int start)

    {

        vector<bool> visited(V, false);

        dfsUtil(start, visited);

        cout<<endl;

    }

};

int main()

{

    int E, V, start\_vertex,u,v;

    cout<<"Enter No of vertices: ";

    cin>>V;

    cout<<"Enter No of Edges: ";

    cin>>E;

    Graph g(V);

    cout<<"Enter Edges (format: Source Destination): "<<endl;

    for(int i = 0; i < E; i++)

    {

        cin>>u>>v;

        g.addEdge(u,v);

    }

    cout<<"Enter start vertex: ";

    cin>>start\_vertex;

    cout<<"BFS Traversal: ";

    g.bfs(start\_vertex);

    cout<<"DFS Traversal: ";

    g.dfs(start\_vertex);

    return 0;

}