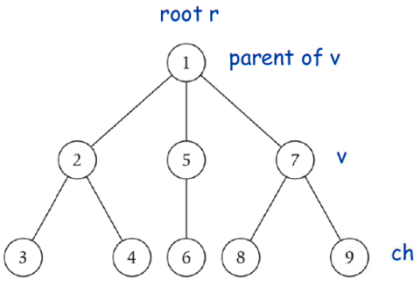
E03-01. Implement algorithms of Breath-First Search(BFS) and give some examples to test it.

Input: a node s, an undirected graph G of n nodes

Output: the layers of nodes, BFS Tree

```cpp
#include <iostream>
#include <vector>
#include <queue>
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);
        adj[v].push_back(u);
    }
    vector<int> bfs(int s) {
        vector<bool> visited(V, false);
        vector<int> layers(V, -1);
        queue<int> q;
        visited[s] = true;
        layers[s] = 0;
        q.push(s);
        while (!q.empty()) {
            int u = q.front();
            q.pop();
            for (int v : adj[u]) {
                if (!visited[v]) {
                    visited[v] = true;
                    layers[v] = layers[u] + 1;
                    q.push(v);
                }
            }
        }
        return layers;
    }
};
```

```cpp
int main() {
    int n, m;
    cout << "请输入图的节点个数和边个数：";
    cin >> n >> m;
    Graph g(n);
    cout << "请输入图的边：" << endl;
    for (int i = 0; i < m; ++i) {
        int u, v;
        cin >> u >> v;
        g.addEdge(u, v);
    }
    int startNode;
    cout << "请输入 BFS 的起始节点：";
    cin >> startNode;
    vector<int> layers = g.bfs(startNode);
    cout << "BFS 树结构如下：" ;
    for (int i = 0; i < n; ++i) {
        cout << "Node " << i << ": " << layers[i] << endl;
    }
    return 0;
}
```



如图，对于该示例 input，运行结果如下：

```
/Users/kkkai/CLionProjects/Algo3/cmake-build-debug/Algo3
请输入图的节点个数和边个数：9 8
请输入图的边：
1 2
1 5
1 7
2 3
2 4
5 6
7 8
7 9
请输入BFS的起始节点：1
BFS树结构如下：Node 0: -1
Node 1: 0
Node 2: 1
Node 3: 2
Node 4: 2
Node 5: 1
Node 6: 2
Node 7: 1
Node 8: 2

进程已结束，退出代码为 0
```

E03-02. Implement algorithms of connected component with BFS and DFS respectively and give some examples to test it.

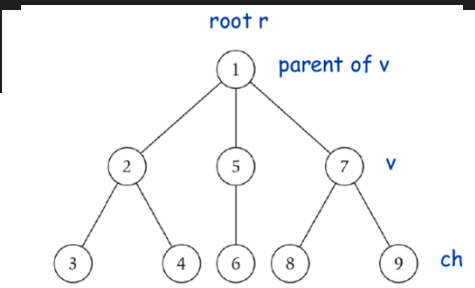
Input: a node s, an undirected graph G of n nodes

Output: the connected component containing *s*.

```
#include <iostream>
#include <vector>
#include <queue>
#include <stack>
using namespace std;
class Graph {
public:
    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);
        adj[v].push_back(u);
    }
    vector<int> bfs(int s) {
        vector<bool> visited(V, false);
        vector<int> layers(V, -1);
        queue<int> q;
        visited[s] = true;
        layers[s] = 0;
        q.push(s);
        while (!q.empty()) {
            int u = q.front();
            q.pop();
            for (int v : adj[u]) {
                if (!visited[v]) {
                    visited[v] = true;
                    layers[v] = layers[u] + 1;
                    q.push(v);
                }
            }
        }
        return layers;
    }
    vector<int> dfs(int s) {
```

```
        vector<bool> visited(V, false);
        vector<int> result;
        stack<int> stk;
        stk.push(s);
        while (!stk.empty()) {
            int u = stk.top();
            stk.pop();
            if (!visited[u]) {
                visited[u] = true;
                result.push_back(u);
                for (int v : adj[u]) {
                    if (!visited[v]) {
                        stk.push(v);
                    }
                }
            }
        }
        return result;
    }
};

int main() {
    int n, m;
    cout << "请输入图的节点个数和边个数：";
    cin >> n >> m;
    Graph g(n);
    cout << "请输入图的边：" << endl;
    for (int i = 0; i < m; ++i) {
        int u, v;
        cin >> u >> v;
        g.addEdge(u, v);
    }
    int startNode;
    cout << "请输入起始节点：";
    cin >> startNode;
    vector<int> layers = g.bfs(startNode);
    vector<int> result = g.dfs(startNode);
    cout << "BFS 求出的连通分量：" ;
    Graph temp=g;
        for (int i=0;i<n+1;i++){
            if(layers[i]!=-1&&i!=startNode) {
                cout << i << ":";
                cout<<endl;
                while(!g.adj[i].empty()) {
```

```
                cout <<"("<< i<< ","<<g.adj[i].front()<<")";
                g.adj[i].erase(g.adj[i].begin());
                cout<<endl;
            }

        }
    }
    cout<<endl;
cout << "DFS 求出的连通分量: " ;
    for(int u:result)
        if(u!=startNode) {
            cout<<endl;
            cout << u << ":";
            while(!temp.adj[u].empty()) {
                cout <<"("<< u<< ","<<temp.adj[u].front()<<")";
                temp.adj[u].erase(temp.adj[u].begin());
                cout<<endl;
            }
        }
cout << "DFS 求出的连通分量: " ;
    for(int u:result)
        if(u!=startNode) {
            cout<<endl;
            cout << u << ":";
            while(!temp.adj[u].empty()) {
                cout <<"("<< u<< ","<<temp.adj[u].front()<<")";
                temp.adj[u].erase(temp.adj[u].begin());
                cout<<endl;
            }
        }
    return 0;
}
```



如图，对于如下示例 input，输出如下：

```
/Users/kkkai/CLionProjects/Algo3/cmake-b
请输入图的节点个数和边个数: 9 8
请输入图的边:
1 2
1 5
1 7
2 3
2 4
5 6
7 8
7 9
请输入起始节点: 1
BFS求出的连通分量: 2:
(2,1)
(2,3)
(2,4)
3:
(3,2)
4:
(4,2)
5:
(5,1)
(5,6)
6:
(6,5)
7:
(7,1)
(7,8)
(7,9)
8:
(8,7)
```

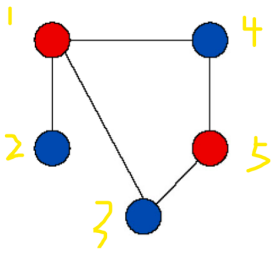
```
8:
(8,7)
9:
(9,7)

DFS求出的连通分量:
7:(7,1)
(7,8)
(7,9)

9:
8:(8,7)

5:(5,1)
(5,6)

6:(6,5)

2:(2,1)
(2,3)
(2,4)

4:(4,2)

3:(3,2)

进程已结束，退出代码为 0
```

E03-03. Implement the algorithms of testing bipartiteness and give some examples to test it.

Input: an undirected graph G of n nodes
Output: “Yes” if G is bipartite graph, or “No” if G is not.

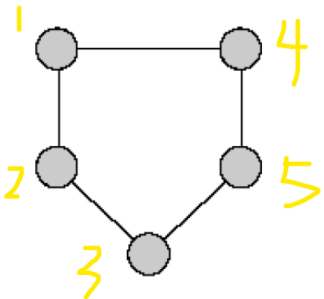
```
#include <iostream>
#include <vector>
#include <queue>
using namespace std;
enum Color { UNCOLORED, RED, BLUE };
class Graph {
public:
    int V;
    vector<vector<int>> adj;
public:
    Graph(int V) : V(V) {
        adj.resize(V);
    }
    void addEdge(int u, int v) {
        adj[u].push_back(v);
        adj[v].push_back(u);
    }
    bool isBipartite() {
        vector<Color> colors(V, UNCOLORED);
        queue<int> q;
        for (int i = 0; i < V; ++i) {
            if (colors[i] == UNCOLORED) {
                colors[i] = RED;
                q.push(i);
                while (!q.empty()) {
                    int u = q.front();
                    q.pop();
                    for (int v : adj[u]) {
                        if (colors[v] == UNCOLORED) {
                            colors[v] = (colors[u] == RED) ? BLUE : RED;
                            q.push(v);
                        } else if (colors[v] == colors[u]) {
                            return false;
                        }
                    }
                }
            }
        }
        return true;
```

```
    }
};
int main() {
    int n, m;
    cout << "输入图的顶点数、边数：";
    cin >> n >> m;
    Graph g(n);
    cout << "输入图的边:" << endl;
    for (int i = 0; i < m; ++i) {
        int u, v;
        cin >> u >> v;
        g.addEdge(u, v);
    }
    if (g.isBipartite())
        cout << "Yes" << endl;
    else
        cout << "No" << endl;
    return 0;
}
```

则对于如下示例： 

，输出如下：

```
输入图的顶点数、边数：5 5
输入图的边：
1 2
1 3
1 4
3 5
4 5
Yes
```

对于如下示例： 

，输出如下：

```
输入图的顶点数、边数：5 5
输入图的边：
1 2
2 3
3 5
4 5
1 4
No
```

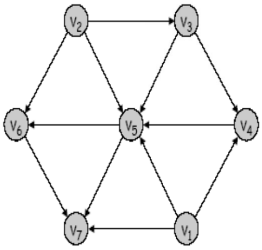
E03-04. Implement the algorithms of the topological order and give some examples to test it.

Input: a directed graph G of n nodes Output: the topological ordering of G

```
#include <iostream>
#include <vector>
#include <queue>
using namespace std;
class Graph {
public:
    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);
    }
    vector<int> topologicalSort() {
        vector<int> indegree(V, 0);
        for (int u = 0; u < V; ++u) {
            for (int v : adj[u]) {
                indegree[v]++;
            }
        }
        queue<int> q;
        for (int u = 0; u < V; ++u) {
            if (indegree[u] == 0) {
                q.push(u);     }
        }
        vector<int> result;
        while (!q.empty()) {
            int u = q.front();
            q.pop();
            result.push_back(u);
            for (int v : adj[u]) {
                indegree[v]--;
                if (indegree[v] == 0) {
                    q.push(v);
                }           }       }
        if (result.size() != V) {
            return vector<int>();       }
        return result;
```

```
    }
};
int main() {
    int n, m;
    cout << "输入有向图的顶点数和边数：";
    cin >> n >> m;
    Graph g(n);
    cout << "输入有向图的边：" << endl;
    for (int i = 0; i < m; ++i) {
        int u, v;
        cin >> u >> v;
        g.addEdge(u, v);
    }
    vector<int> topologicalOrder = g.topologicalSort();
    if (topologicalOrder.empty()) {
        cout << "有向图含环，无法构成拓扑序列" << endl;
    } else {
        cout << "该有向图的一种可能拓扑序列如下：";
        for (int node : topologicalOrder) {
            cout << node << " ";
        }
        cout << endl;
    }
    return 0;
}
```

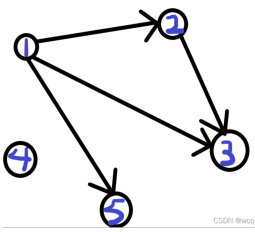
则对于如下输入：



输出如下：

```
输入有向图的顶点数和边数：7 12
输入有向图的边：
2 6
2 5
2 3
3 5
5 6
3 4
4 5
6 7
5 7
1 4
1 5
1 7
有向图含环，无法构成拓扑序列
```

对于如下输入：



输出如下

```
输入有向图的顶点数和边数：5 4
输入有向图的边：
1 2
1 3
1 5
2 3
该有向图的一种可能拓扑序列如下：1 4 2 5 3
```