Cycle All Around

Given an undirected simple graph, find whether it is possible to traverse the whole graph and return to original position without visiting any vertex twice. If it is not possible to traverse it then print ‘‘No’’ as output. Otherwise you have to print a valid node traversal. You have to visit each vertex exactly once.

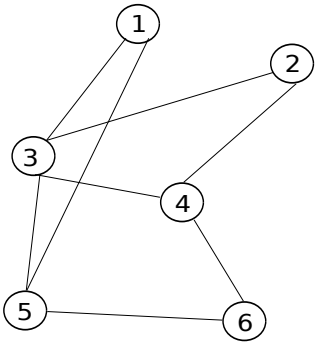

Figure 1: An example graph

The nodes are numbered 1...n. In the example graph shown in figure 1, there are 6 nodes. Hence, $n = 6$. There is several valid traversal of the nodes which forms a Hamiltonian cycle for example 1 3 2 4 6 5 1.

Input Specification

First line of input consists of two integers n and m. $1 \leq n \leq 10$ and $1 \leq m \leq n(n+1)/2$. First integer denotes number of vertices in the graph, and the second integer denotes number of edges. The next m lines each contain two integers u, v which means there is an edge u, v in the graph. The input file may contain description of multiple graphs. Each of the graph descriptions will be separated by a single blank line. You should stop taking inputs when 0 is input for both n and m.

Output Specification

For each of the graphs in the input file, print one line which contains a valid node traversal as described above. Separate the vertex numbers using a single space. Check the sample output section for clarification.

Sample Input

```
5 8
1 2
1 4
4 3
2 3
3 1
2 4
1 5
4 5

4 6
1 2
2 3
3 4
4 1
1 3
2 4

6 8
1 3
1 5
2 3
2 4
3 4
3 5
5 6
4 6

6 8
1 3
1 5
2 3
2 4
3 4
3 5
5 6
3 6

0 0
```

Sample Output

```
1 3 2 4 5 1
1 2 3 4 1
1 2 3 4 5 6 1
No
```

Remarks

Graph Representation You must use an adjacency list representation for the graph. Your graph should have an associated data structure, with which you will be able to find all the neighbors of a node u in $O(deg[u])$ time. Try to follow the adjacency represenstation style shown in the class. However, you are free to use `STL` data structures, if you like.

Algorithm You may use the idea of `DFS` (Depth First Search) algorithm to complete this assignment. However, just writing the `DFS` code exactly like book may not work. You need to make some adjustments in the node coloring sequence and the node visiting decision. Usually in DFS, when we visit a node we mark it as visited. And never visit it again. But if we visit a vertex and come to a fail to find any path, then when we return from that vertex and try another path, that vertex will not be visited again. However, as the objective here is to visit every vertex exactly once, you should keep a track of the already visited verteices on current path (not those visited once), and should not visit them again.

Sketch of a Solution Start with node `1` or any other appropriate node. From this node, visit other nodes in a depth first manner not visiting same edge twice. At each step, keep track of the vertex you are visiting. If you have completed visiting all the vertex, then print the sequence as you completed. However, if you are stuck, *i.e.* you are in a vertex whose all neighbors are visited, but still haven't visited all the vertices , then go back to the last visited vertex, unvisit the last vertex you visited from there, choose a different neighbour and proceed from there. If you can't choose any different neighbour from there, then go back one step more and repeat.

Extra Credit Additionally, you can print all the node sequences that completes the Hamiltonian cycle of 3 marks bonus. If there is more than one Hamiltonian cycle in the graph, you should print only the node sequences starting with `1`. If you can code the bonus portion, you should make it a separate program, and please keep the original file intact which prints the output exactly as specified in this problem specification.

Input Output Explanation In the sample input given, there are three graphs. The first graph starts with the line `5 8` and ends with the line `4 5`. Output for that graph is the first line in the sample output. The second graph starts with `4 6` in the sample input file, and the output for that graph is `No` as shown the second line in the sample output. Output for third graph is the third line of the sample output file. However, the last line of input is `0 0`. Obviously, you should not process this as a graph, rather just exit from your program. So your code should take input of graphs in a loop and process them. The third sample is the example shown in figure 1.

```
while(scanf("%d%d",&n,&m)==2)
{
    if(n==0 && m==0) break;

    // some initialization here...

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

```
    {
        scanf("%d%d",&u,&v);

        // insert the edge (u,v) in the adjacency list
        // ........
    }

    // .... process and print output
}
```

Tanaeem M Moosa
tanaeem.moosa@csebuet.org