Draw At Once

Given an undirected simple graph, find whether it is possible to draw it without lifting the pen. If it is not possible to draw it without lifting the pen from the paper, then print ``No'' as output. Otherwise you have to print a valid node traversal which completes the drawing. You may need to visit a vertex more than once, but you can't visit an edge twice *i.e.* you have to find an eulerian circuit / trail in the graph.

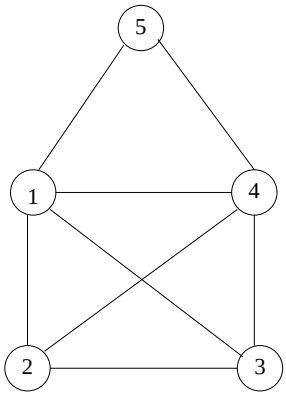

Figure 1: An example graph

The nodes are numbered 1...n. In the example graph shown in figure 1, there are `5` nodes. Hence, $n = 5$. There is several valid traversal of the nodes which can draw the graph without lifting the pen from the paper, one of them is `2 1 5 4 1 3 4 2 3`.

Input Specification

First line of input consists of two integers n and m. $1 \leq n \leq 100$ and $1 \leq m \leq 4950$. 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 6
1 2
2 3
3 4
4 5
5 6
1 6

0 0
```

Sample Output

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

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. For example, in the book algorithm for `DFS`, initially all the nodes are `WHITE`, and they are colored `GRAY` as discovered, and already discovered nodes are not visited again. However, as the objective here is to visit every edge exactly once, you should keep a track of the already visited edges, 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 edge you are visiting. If you have completed visiting all the edges, then print the sequence as you completed. However, if you are stuck, *i.e.* you have visited some of the edges, but can't proceed further, then go back to the last visited vertex, unvisit the last edge you visited from there, choose a different one and proceed from there. If you can't choose any different edge from there, then go back one step more and repeat. For example, in the graph shown in figure 1, Inside your algorithm, you may need to find whether there is an (u, v) edge. You can keep an additional representation for the graph using adjacency matrix to answer that query $O(1)$ time.

Extra Credit Additionally, you can print all the node sequences that completes the drawing for a maximum of 3 marks bonus. If there is an eulerian circuit in the graph, you should print only the node sequences starting with `1`. If there is just an eulerian trail but no circuit, you should print the node sequences which start with the minimum of the two odd degree vertices. 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.

```
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
}
```

Md. Tanvir Al Amin
tanviralamin@gmail.com