#Problem 1: Cycle Finding and Printing

Description: In this problem you will be given an undirected unweighted graph G. You have to determine if the graph contains any cycle or not.

In the first line you will be given two integers V and E denoting the number of vertices and edges of this graph respectively. All the vertices will have ids between 1 to V. In the following E lines, you will get the information about all the edges. In the i^{th} line you will have two integers x (1<=x<=V) and y (1<=y<=V) which denotes there is an undirected edge between x and y.

You need to write a program that will detect if the graph contains any cycle or not. If not found print “NO” (without quote). But if the graph contains any cycle, you need to print “YES” (without quote). In the following line you need to print the vertices which make the cycle maintaining the ascending order of the vertices’ ids. No vertex will be repeated in the printing. In this problem, to have a cycle, you will need at least three nodes. In this problem you can safely assume that the given graph will contain a single cycle or not. Print the vertices separated by a single space.

Limits
1<=V<=100000, 1<=|E|<=100000

Test Cases:

Input	Output
4 3 1 2 1 3 1 4	NO
4 4 1 2 1 3 1 4 2 4	YES 1 2 4
10 10 1 2 1 5 2 4 3 4 3 5 4 8 6 10 6 7 8 10 9 10	YES 1 2 3 4 5

10 9 1 2 2 3 2 4 3 5 4 9 6 8 6 7 7 10 8 9	No

#Problem 2: Maximal Connected Component in an Undirected Graph

Description: In this problem you will be given an undirected unweighted graph G. You have to discover the maximal connected component from this graph. A connected component C_1 is larger compared to the other connected component C_2 iff C1 contains more vertices than C2.

In the first line you will be given two integers V and E denoting the number of vertices and edges of this graph respectively. All the vertices will have ids between 1 to V. In the following E lines, you will get the information about all the edges. In the i^{th} line you will have two integers x (1<=x<=V) and y (1<=y<=V) which denotes there is an undirected edge between x and y.

You need to write a program that will detect the maximal connected component from the given graph as input. There will be a single line of output for each test case containing the number of vertices that make the maximal connected component.

Limits
1<=V<=1000, 1<=|E|<=1000000

Test Cases:

Input	Output
5 6 1 2 2 3 3 4 4 5 1 4 1 5	5
15 15 1 8 1 7 2 10 2 3	10

3 9 4 7 4 9 5 6 5 8 6 10 11 14 11 15 12 13 12 14 13 15	

#Problem 3: Topological Sorting in Ascending order

Description: In this problem, you will be given a directed unweighted graph G and a set of partial ordered tasks, 3 -> 7 where 3 needs to be completed before 7. Tasks will be represented through node numbers. You need to calculate a valid topological ordering among the tasks. During printing the tasks, you need to maintain an ascending order among the tasks that have no ordered dependencies among them. It means that if task 2 and 7 have no ordered dependency between them, then 2 needs to be completed first (or printed first) before 7.

In the first line you will be given two integers V and E denoting the number of vertices and edges of this graph respectively. All the vertices will have ids between 1 to V. In the following E lines, you will get the information about all the edges. In the i^{th} line you will have two integers x (1<=x<=V), y (1<=y<=V) which denotes there is a directed edge from x to y.

There will be V lines of output. Each line denoting a single node number representing a valid node or task topologially sorted. If i < j, then the i^{th} task needs to be done before the j^{th} task.

Limits
1<=V<=100000, 1<=|E|<=100000

Test Cases:

Input	Output
5 5 1 2 1 3 1 4 2 5 3 5	1 2 3 4 5
8 9 2 1	3 8

3 7 4 5 5 2 6 2 7 2 8 4 8 7 8 6	4 5 6 7 2 1
20 19 3 15 7 14 8 10 9 8 9 13 9 19 9 7 9 3 9 16 9 17 9 2 9 4 9 5 9 20 9 1 13 11 16 12 17 18 19 6	9 1 2 3 4 5 7 8 10 13 11 14 15 16 12 17 18 19 6 20