Trajan Algorithm

Input file:	`standard input`
Output file:	`standard output`
Time limit:	2 seconds
Memory limit:	1024 megabytes

Recall the *Tarjan Algorithm* for calculating the articulation points (also called the cut vertices) in an undirected graph, whose pseudo-code is shown on the left.

Algorithm 1 The Correct Tarjan Algorithm

```
1: function TARJAN(i, d)
2:     visited[i] ← true
3:     depth[i] ← d
4:     low[i] ← d
5:     childCount ← 0
6:     isArticulation ← false
7:
8:     for each j in adj[i] do
9:         if j = parent[i] then
10:            continue
11:        end if
12:        if not visited[j] then
13:            parent[j] ← i
14:            TARJAN(j, d + 1)
15:            childCount ← childCount + 1
16:            if low[j] ≥ depth[i] then
17:                isArticulation ← true
18:            end if
19:            low[i] ← min(low[i], low[j])
20:        else
21:            low[i] ← min(low[i], depth[j])
22:        end if
23:    end for
24:    if (parent[i] ≠ null and isArticulation)
   or (parent[i] = null and childCount > 1)
   then
25:        Output i as articulation point
26:    end if
27: end function
```

Algorithm 2 The Incorrect Trajan Algorithm

```
1: function TRAJAN(i, d)
2:     visited[i] ← true
3:     depth[i] ← d
4:     low[i] ← d
5:     childCount ← 0
6:     isArticulation ← false
7:
8:     for each j in adj[i] do
9:         if j = parent[i] then
10:            continue
11:        end if
12:        if not visited[j] then
13:            parent[j] ← i
14:            TRAJAN(j, d + 1)
15:            childCount ← childCount + 1
16:            if low[j] ≥ depth[i] then
17:                isArticulation ← true
18:            end if
19:        end if
20:        low[i] ← min(low[i], low[j])
21:
22:
23:    end for
24:    if (parent[i] ≠ null and isArticulation)
   or (parent[i] = null and childCount > 1)
   then
25:        Output i as articulation point
26:    end if
27: end function
```

Note that in the pseudo code, $adj[i]$ is the array containing all vertices adjacent to vertex i, *parent* is an array in which the initial value of each element is null, and *visited* is an array in which the initial value of each element is false.

Some students may have a hard time understanding this algorithm, especially from line 19 to line 22. They can't understand why the *Tarjan Algorithm* needs to calculate the value of $low[i]$ differently, and they may bear an incorrect version of the algorithm in mind. Let's call this incorrect version of the algorithm the *Trajan Algorithm*, which is shown on the right. Notice that the *Tarjan* and the *Trajan* algorithms are almost the same, except for the function names and the 4 lines from line 19 to line 22.

Given an undirected connected graph with n vertices and m edges, for each vertex u determine if there exists a vertex v and a 2-dimensional array *adj* (*adj* must represent the given graph) such that TARJAN(v, 0) and TRAJAN(v, 0) produce different results on vertex u (that is to say, the *Tarjan Algorithm* states that vertex u is an articulation point while the *Trajan Algorithm* does not, or vice versa).

Input

There are multiple test cases. The first line of the input contains an integer T ($1 \leq T \leq 10^4$) indicating the number of test cases. For each test case:

The first line contains two integers n and m ($2 \leq n \leq 10^5$, $1 \leq m \leq 10^5$) indicating the number of vertices and edges in the given graph.

For the following m lines, the i-th line contains two integers u_i and v_i ($1 \leq u_i, v_i \leq n$) indicating that there is an edge connecting vertex u_i and v_i in the graph.

It's guaranteed that the given graph is connected and there are no self loops or multiple edges. Also, neither the sum of n nor the sum of m of all test cases will exceed 10^6.

Output

For each test case output one line containing some integers separated by one space in increasing order. Each integer is a vertex that the *Tarjan* and the *Trajan* algorithm may produce different results on. If there are no such vertices, output `Empty` in one line instead.

Example

standard input	standard output
2 9 12 1 3 2 3 1 2 3 6 6 9 9 3 4 6 5 6 4 5 7 9 8 9 7 8 2 1 1 2	3 6 9 Empty

Note

For the first sample test case:

- For $u = 3$ consider $v = 1$ and $adj = [[3, 2], [3, 1], [1, 2, 6, 9], [6, 5], [6, 4], [4, 5, 3, 9], [9, 8], [9, 7], [7, 8, 3, 6]]$.
- For $u = 6$ consider $v = 4$ and $adj = [[2, 3], [1, 3], [9, 6, 2, 1], [6, 5], [6, 4], [4, 5, 3, 9], [9, 8], [9, 7], [7, 8, 3, 6]]$.
- For $u = 9$ consider $v = 7$ and $adj = [[2, 3], [1, 3], [9, 6, 2, 1], [5, 6], [4, 6], [9, 3, 5, 4], [9, 8], [9, 7], [7, 8, 3, 6]]$.

Note that all *adj*s in the explanation are 1-indexed.