"Name and Class Year:
Course to be Covered: (Course Number and Title)
Reason for covering the course independently:"
`Hamilton College Application for Independent Coverage of Course Work`

Definitions

In this problem, a *graph* is a set of n vertices together with a set of m edges, where an *edge* is an unordered pair of different vertices (edges are undirected). The two vertices that comprise an edge are said to be that edge's *endpoints*. A *vertex cover* of a given graph G is a subset C of its vertices, such that each edge of G has at least one of its endpoints in C. An *independent set* of a given graph G is a subset S of its vertices, such that no edge of G has both of its endpoints in S.

The problem of finding a *minimum vertex cover* (that is, a vertex cover of the smallest possible size) for any graph is NP-hard. The problem of finding a *maximum independent set* of any graph is also NP-hard. That is a formal way of saying that no one knows whether there exists an algorithm that runs in time polynomial in n and solves any one of the two problems.

We want to define a class of problems that are even harder than the NP-hard problems. We are going to call them "Double NP-hard"! Your job is to solve the first Double NP-hard problem.

Problem

Given a graph G, find a subset C of its vertices that is both a *minimum vertex cover* and a *maximum independent set*.

Input

The first line of input gives the number of cases, N. N test cases follow. Each one starts with two lines containing n ($0 \leq n \leq 1000$) and m ($0 \leq m \leq 100000$) as above. The next m lines will each describe an edge of G as a pair of different vertices, which are numbered from 1 to n.

Output

For each test case, output one line containing '`Case #`x`:`' followed by either '`Impossible`' if there is no answer or the size k of the set C. In the latter case, on the next line, print the k vertices of C in increasing order, separated by spaces. If there are multiple answers, print the lexicographically smallest one.

Sample Input

```
4
2
1
1 2
0
0
10
0
4
4
1 2
2 3
3 4
4 1
```

Sample Output

```
Case #1: 1
1
Case #2: 0

Case #3: Impossible
Case #4: 2
1 3
```