A - 23 Out of 5

Source file name: `out.py`
Time limit: 1 second

Your task is to write a program that can decide whether you can find an arithmetic expression consisting of five given numbers a_i ($1 \leq i \leq 5$) that will yield the value 23. For this problem we will only consider arithmetic expressions of the following from:

$$\left(\left(\left(\left(a_{\pi(1)} \oplus_1 a_{\pi(2)}\right) \oplus_2 a_{\pi(3)}\right) \oplus_3 a_{\pi(4)}\right) \oplus_4 a_{\pi(5)}\right)$$

where $\pi : \{1,2,3,4,5\} \to \{1,2,3,4,5\}$ is a bijective function and $\oplus_i \in \{+,-,*\}$, with $1 \leq i \leq 4$.

Input

The input consists of 5-tuples of positive integers, each between 1 and 50. Input is terminated by a line containing five zero's. This line should not be processed.

The input must be read from standard input.

Output

For each 5-tuple output "`Possible`" (without the quotes) if there is an arithmetic expression (as described above) that yields 23. Otherwise, output "`Impossible`".

The output must be written to standard output.

Sample Input	Sample Output
`1 1 1 1 1`	`Impossible`
`1 2 3 4 5`	`Possible`
`2 3 5 7 11`	`Possible`
`0 0 0 0 0`	

B - Beverages

Source file name: `beverages.py`
Time limit: 1 second

Dilbert has just finished college and decided to go out with friends. Dilbert has some strange habits and thus he decided to celebrate this important moment of his life drinking a lot. He will start drinking beverages with low alcohol content, like beer, and then will move to a beverage that contains more alcohol, like wine, until there are no more available beverages. Once Dilbert starts to drink wine he will not drink beer again, so the alcohol content of the beverages never decreases with the time. You should help Dilbert by indicating an order in which he can drink the beverages in the way he wishes.

Input

Each test case starts with $1 \leq N \leq 100$, the number of available beverages. Then will follow N lines, informing the name of each beverage, a name has less than 51 characters and has no white spaces. Then there is another line with an integer $0 \leq M \leq 200$ and M lines in the form B_1 B_2 will follow, indicating that beverage B_2 has more alcohol that beverage B_1, so Dilbert should drink beverage B_1 before he starts drinking beverage B_2. Be sure that this relation is transitive, so if there is also a relation B_2 B_3 you should drink B_1 before you can drink B_3. There is a blank line after each test case. The input is terminated by end of file (EOF).

The input must be read from standard input.

Output

For each test case you must print the message:

```
Case #C: Dilbert should drink beverages in this order: B1 B2 ... BN.
```

where C is the number of the test case, starting from 1, and B_1 $B_2 \ldots B_N$ is a list of the beverages such that the alcohol content of beverage B_{i+1} is not less than the alcohol content of beverage B_i. In the case there is no relation between two beverages Dilbert should start drinking the one that appears first in the input. After each test case you must print a blank line.

The output must be written to standard output.

Sample Input

```
3
vodka
wine
beer
2
wine vodka
beer wine

5
wine
beer
rum
apple-juice
cachaca
6
beer cachaca
apple-juice beer
apple-juice rum
beer rum
beer wine
wine cachaca
```

Sample Output

```
Case #1: Dilbert should drink beverages in this order: beer wine vodka.

Case #2: Dilbert should drink beverages in this order: apple-juice beer wine rum cachaca.
```

C - Traffic Flow

Source file name: `flow.py`
Time limit: 1 second

A city has n intersections and m bidirectional roads connecting pairs of intersections. Each road has a certain traffic flow capacity, measured in cars per minute. There is a path from every intersection to every other intersection along some sequence of roads. The road maintenance department is over budget and needs to close as many roads as possible without disconnecting any intersections. They want to do it in such a way that the minimum capacity among all of the remaining roads is as large as possible.

Input

The first line of input gives the number of cases, N. Then N test cases follow. Each one starts with a line containing n ($0 < n \leq 100$) and m ($0 < m \leq 10000$). The next m lines will describe the m roads, each one using 3 integers u, v, and c ($0 \leq u, v < n$), ($0 < c \leq 1000$); u and v are the endpoints of the road and c is its capacity.

The input must be read from standard input.

Output

For each test case, output one line containing "`Case #x:`" followed by the capacity of the minimum-capacity remaining road.

The output must be written to standard output.

Sample Input	Sample Output

`2`
`2 3`
`0 1 10`
`0 1 20`
`0 0 30`
`4 5`
`0 1 1`
`3 1 2`
`1 2 3`
`2 3 4`
`0 2 5`
	`Case #1: 20`
	`Case #2: 3`

D - Is There a Second Way?

Source file name: `second.py`
Time limit: 1 second

Nasa, being the most talented programmer of his time, can't think things to be so simple. Recently all his neighbors have decided to connect themselves over a network (actually all of them want to share a broadband internet connection `:-)`). But he wants to minimize the total cost of cable required as he is a bit fastidious about the expenditure of the project. For some unknown reasons, he also wants a second way left. I mean, he wants to know the second best cost (if there is any which may be same as the best cost) for the project. I am sure, he is capable of solving the problem. But he is very busy with his private affairs(?) and he will remain so. So, it is your turn to prove yourself a good programmer. Take the challenge (if you are brave enough) ...

Input

Input starts with an integer $t \leq 100$ which denotes the number of test cases to handle. Then follow t datasets where every dataset starts with a pair of integers v $(1 \leq v \leq 100)$ and e $(0 \leq e \leq 200)$; v denotes the number of neighbors and e denotes the number of allowed direct connections among them. The following e lines contain the description of the allowed direct connections where each line is of the form '*start end cost*', where *start* and *end* are the two ends of the connection and *cost* is the cost for the connection. All connections are bi-directional and there may be multiple connections between two ends.

The input must be read from standard input.

Output

There may be three cases in the output: (1) No way to complete the task, (2) There is only one way to complete the task, and (3) There are more than one way. Output '`No way`' for the first case, '`No second way`' for the second case, and an integer c for the third case, where c is the second best cost. Output for a case should start in a new line.

The output must be written to standard output.

Sample Input	Sample Output

```
4
5 4
1 2 5
3 2 5
4 2 5
5 4 5
5 3
1 2 5
3 2 5
5 4 5
5 5
1 2 5
3 2 5
4 2 5
5 4 5
4 5 6
1 0
```
	```
	Case #1 : No second way
	Case #2 : No way
	Case #3 : 21
	Case #4 : No second way
	```