2011-0920 UM Online Practice Programming Contest
September 20, 2011

Sponsored by:
University of Michigan

Rules:

1. There are a few questions to be solved any time during the tryouts contest.
2. All questions require you to read the test data from standard input and write results to standard output. You cannot use files for input or output. Additional input and output specifications can be found in the General Information Sheet.
3. The allowed programming languages are C, C++ and Java.
4. All programs will be re-compiled prior to testing with the judges' data.
5. Non-standard libraries cannot be used in your solutions. The Standard Template Library (STL) and C++ string libraries are allowed. The standard Java API is available, except for those packages that are deemed dangerous by contestant officials (e.g., that might generate a security violation).
6. Programming style is not considered in this contest. You are free to code in whatever style you prefer. Documentation is not required.
7. All communication with the judges will be handled by the PC2 environment.
8. Judges' decisions are to be considered final. No cheating will be tolerated.

Problem A: Pass Feil!

Todd Feil is one of the ICPC Judges. Really. Watch out, as he will try to fail you now, because one day he discovered that his class has an unusual amount of students named Bartholomew. To find out just exactly how many Barts are in his class, he wrote down each student's name on a sheet of paper and got the following list:

John, Bart, Michael, Bart, John, Bart, Peter, Eugene, Dave, David, Bart

Then, he counted up the number of Barts he wrote down, and he was right! There were 4 Bart students, more Barts than students with any other name in the class. "Hmm", said Professor Feil to himself hardly trying to suppress evil cackling, "It will be most interesting to find out the most popular student name in the entire school and also, how many students have that name". The longer he thought about this the more he wanted to know the answer. He knew it will be a lot of work, since school has more than 2,000 people! The more he thought about it the more tedious it became. Professor Feil realized that he fell short of his own expectations. Recalling that his class has a very bright student – you! – he now seeks your help. Professor wants you to write a program that takes a list of names, and figures out how many students on campus have the most popular name and what that name is. And if you don't come up with the answer, he will surely fail you and you will have to put up with Prof. Feil's smugly-triumphal smirk for the rest of the semester. Clearly, you better get right on this.

Input

Input will consist of multiple cases. Each line will contain an integer n between 1 and 2,500 inclusively, where n the number of student names to follow, followed by n student names separated by a space. Each name will be no longer than 20 characters and consist of characters 'a' through 'z' with the first letter capitalized. Last line will start with 0, and should not be processed. You are guaranteed that there will be a unique most popular name for every case.

Output

For each line of input professor expects you to print out the most popular name in the list, followed by the number of how many students have that name, with a space in between. The name should be capitalized as it was in the original list.

Sample Input

```
11 John Bart Michael Bart John Bart Peter Eugene Dave David Bart
10 Mike Mark Dennis Raiden Smoke Belokk Goro Kevin Mike Andy
0
```

Sample Output

```
Bart 4
Mike 2
```

Problem B: Rotate This!

In the exciting game of Join-**K**, red and blue pieces are dropped into an **N**-by-**N** table. The table stands up vertically so that pieces drop down to the bottom-most empty slots in their column. For example, consider the following two configurations:

```
- Legal Position -    - Illegal Position -

.......                 .......
.......                 .......
.......                 .......
....R..                 .......
...RB..          Bad -> ..BR...
..BRB..                 ...R...
.RBBR..                 .RBBR..
```

In these pictures, each '.' represents an empty slot, each 'R' represents a slot filled with a red piece, and each 'B' represents a slot filled with a blue piece. The left configuration is legal, but the right one is not. This is because one of the pieces in the third column (marked with the arrow) has not fallen down to the empty slot below it.

A player wins if they can place at least **K** pieces of their color in a row, either horizontally, vertically, or diagonally. The four possible orientations are shown below:

```
  - Four in a row -

R   RRRR    R   R
R         R     R
R        R       R
R       R         R
```

In the "Legal Position" diagram at the beginning of the problem statement, both players had lined up two pieces in a row, but not three.

As it turns out, you are right now playing a very exciting game of Join-**K**, and you have a tricky plan to ensure victory! When your opponent is not looking, you are going to rotate the board 90 degrees clockwise onto its side. Gravity will then cause the pieces to fall down into a new position as shown below:

```
- Start -    - Rotate -    - Gravity -

.......       .......       .......
.......       R......       .......
.......       BB.....       .......
...R...       BRRR...       R......
...RB..       RBB....       BB.....
..BRB..       .......       BRR....
.RBBR..       .......       RBBR...
```

Unfortunately, you only have time to rotate once before your opponent will notice.

All that remains is picking the right time to make your move. Given a board position, you should determine which player (or players!) will have **K** pieces in a row after you rotate the board clockwise and gravity takes effect in the new direction. Note that:

- You can rotate the board only once.
- Assume that gravity only takes effect after the board has been rotated completely.
- Only check for winners after gravity has finished taking effect.

Input

The first line of the input gives the number of test cases, **T,** 1 ≤ **T** ≤ 100. **T** test cases follow, each beginning with a line containing the integers **N,** 3 ≤ **N** ≤ 50 and **K,** 3 ≤ **K** ≤ **N**. The next **N** lines will each be exactly **N** characters long, showing the initial position of the board, using the same format as the diagrams above.

The initial position in each test case will be a legal position that can occur during a game of Join-**K**. In particular, neither player will have already formed **K** pieces in a row.

Output

For each test case, output one line containing "Case #x: y", where x is the case number starting from 1, and y is one of "Red", "Blue", "Neither", or "Both". Here, y indicates which player or players will have **K** pieces in a row after you rotate the board.

Sample Input

```
4
7 3
.......
.......
.......
...R...
...BB..
..BRB..
.RRBR..
6 4
......
......
.R...R
.R..BB
.R.RBR
RB.BBB
4 4
R...
BR..
BR..
BR..
3 3
B..
RB.
RB.
```

Sample Output

```
Case #1: Neither
Case #2: Both
Case #3: Red
Case #4: Blue
```

Problem C: FreeCell Stats

Mark played **D** (**D** > 0) games of FreeCell today. Each game of FreeCell ends in one of two ways – Mark either wins, or loses. Mark has been playing for many years, and has so far played **G** games in total (naturally, **G** ≥ **D**).

At the end of the day, Mark looks at the game statistics to see how well he played. It turns out that he won exactly $\mathbf{P_D}$ percent of the **D** games today, and exactly $\mathbf{P_G}$ percent of **G** total games he had ever played. Miraculously, there is no rounding necessary -- both percentages are exact! Unfortunately, Mark does not remember the exact number of games he played today (**D**), or the exact number of games that he played in total (**G**). He does know that he could not have played more than **N** games today (**D** ≤ **N**).

Are the percentages displayed possible, or is the game statistics calculator broken?

Input

The first line of the input gives the number of test cases, **T,** 1 ≤ **T** ≤ 2000. **T** lines follow. Each line contains 3 integers – **N,** 1 ≤ **N** ≤ 10^{15}, $\mathbf{P_D}$, 0 ≤ $\mathbf{P_D}$ ≤ 100 and $\mathbf{P_G}$, 0 ≤ $\mathbf{P_G}$ ≤ 100.

Output

For each test case, output one line containing "Case #x: y", where x is the case number (starting from 1) and y is either "Possible" or "Broken".

Sample Input

```
3
1 100 50
10 10 100
9 80 56
```

Sample Output

```
Case #1: Possible
Case #2: Broken
Case #3: Possible
```

Sample Explanation

In Case #3, Mark could have played 5 games today (**D** = 5) and 25 games in total (**G** = 25), and won 4 games today (80% of 5) and 14 games in total (56% of 25). Note: O(N) algorithm might be slow.

Problem D: Milkshakes

You've decided to own a milkshake shop. There are N different milkshake flavors that you can prepare for eager customers. Each flavor can be prepared as "malted" or "unmalted". Test time! How many milkshakes can you make? That's right, you can make a total of 2N different types of milkshakes!

Each of your customers has a set of milkshake types that they like, and they will be satisfied if you have at least one of those types prepared by the time they walk through the door. At most one of those types a customer will like is a malted flavor. You want to make N batches of milkshakes, so that:

- There is exactly one batch for each flavor of milkshake, and it is either malted or unmalted.
- For each customer, you make at least one milkshake type that they like.
- The minimum possible number of batches are malted.

Find out, whether it is possible to satisfy all of your customers, given these constraints, and if, what milkshake types you should make. If it is possible to satisfy all your customers, there will be only one answer which minimizes the number of malted batches.

Input

One line containing an integer **C** (1 <= C <= 5), the number of test cases in the input file.
For each test case, there will be:

- One line containing the integer **N** (1 <= N <= 2,000), the number of milkshake flavors.
- One line containing the integer **M** (1 <= M <= 2,000), the number of customers.
- **M** lines, one for each customer, each containing:
    - An integer **T** >= 1, the number of milkshake types the customer likes, followed by
    - **T** pairs of integers "**X Y**", one for each type the customer likes, where **X** is the milkshake flavor between **1** and **N** inclusive, and **Y** is either 0 to indicate unmalted, or 1 to indicated malted. Note that:
        - No pair will occur more than once for a single customer.
        - Each customer will have at least one flavor that they like (T >= 1).
        - Each customer will like at most one malted flavor. (At most one pair for each customer has Y = 1).
- All of these numbers are separated by single spaces.

Output

- **C** lines, one for each test case in the order they occur in the input file, each containing the string "Case #**X**: " where **X** is the number of the test case, starting from 1, followed by:
    - The string "**IMPOSSIBLE**", if the customers' preferences cannot be satisfied; **OR**
    - **N** space-separated integers, one for each flavor from **1** to **N**, which are 0 if the corresponding flavor should be prepared unmalted, and 1 if it should be malted.

Sample Input

```
2
5
3
1 1 1
2 1 0 2 0
1 5 0
1
2
1 1 0
1 1 1
```

Sample Output

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

Sample Explanation

In the first case, you must make flavor #1 malted, to satisfy the first customer. Every other flavor can be unmalted. The second customer is satisfied by getting flavor #2 unmalted, and the third customer is satisfied by getting flavor #5 unmalted. In the second case, there is only one flavor. One of your customers wants it malted and one wants it unmalted. You cannot satisfy them both.

Problem E: Watch Tower

In a one dimensional mountainous landscape you want to build a watchtower. This watchtower must fulfill one condition: it must be possible to oversee the whole landscape from it. The landscape is described by pair of integers **positions** and **heights** and the intermediate heights are obtained by interpolation

You are allowed to build the watchtower on the landscape wherever you like. To save expenses, you want to build it at a position such that the necessary height of the watchtower needed to oversee the whole landscape is minimized. Figure it out this minimal height!

Input

The data contains multiple test cases and each case starts with **N** (2 <= N <= 50) representing the number of pairs. Then following N lines have an integer pair "positions_i heights_i" for ith line and positions_i are sorted in strctly ascending order (0 <= positions and heights <= 1,000,000). The last test case ends with N = 0 and you must ignore this case.

Output

Print out the minimal height described above. For the precision issue, relative or absolute error less than 1e-6 is considered to be right.

Sample Input

```
6
1 1
2 2
4 2
5 4
6 2
7 1
4
10 0
20 10
49 10
59 0
0
```

Sample Output

```
1.000000
14.500000
```