Problem A. Magic Trick

Small input
6 points

Solve A-small

Note: To advance to the next rounds, you will need to score 25 points. Solving just this problem will not give you enough points.

Problem

Recently you went to a magic show. You were very impressed by one of the tricks, so you decided to try to figure out the secret behind it!

The magician starts by arranging 16 cards in a square grid: 4 rows of cards, with 4 cards in each row. Each card has a different number from 1 to 16 written on the side that is showing. Next, the magician asks a volunteer to choose a card, and to tell him which row that card is in.

Finally, the magician arranges the 16 cards in a square grid again, possibly in a different order. Once again, he asks the volunteer which row her card is in. With only the answers to these two questions, the magician then correctly determines which card the volunteer chose. Amazing, right?

You decide to write a program to help you understand the magician's technique. The program will be given the two arrangements of the cards, and the volunteer's answers to the two questions: the row number of the selected card in the first arrangement, and the row number of the selected card in the second arrangement. The rows are numbered 1 to 4 from top to bottom.

Your program should determine which card the volunteer chose; or if there is more than one card the volunteer might have chosen (the magician did a bad job); or if there's no card consistent with the volunteer's answers (the volunteer cheated).

Solving this problem

Usually, Google Code Jam problems have 1 Small input and 1 Large input. This problem has only **1 Small input**. Once you have solved the Small input, you have finished solving this problem.

Input

The first line of the input gives the number of test cases, **T**. **T** test cases follow. Each test case starts with a line containing an integer: the answer to the first question. The next 4 lines represent the first arrangement of the cards: each contains 4 integers, separated by a single space. The next line contains the answer to the second question, and the following four lines contain the second arrangement in the same format.

Output

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

If there is a single card the volunteer could have chosen, y should be the number on the card. If there are multiple cards the volunteer could have chosen, y should be "Bad magician!", without the quotes. If there are no cards consistent with the volunteer's answers, y should be "Volunteer cheated!", without the quotes. The text needs to be exactly right, so consider copying/pasting it from here.

Limits

$1 \leq \mathbf{T} \leq 100$.
$1 \leq$ both answers ≤ 4.
Each number from 1 to 16 will appear exactly once in each arrangement.

Sample

Input	Output
3 2 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 3 1 2 5 4 3 11 6 15 9 10 7 12 13 14 8 16	Case #1: 7 Case #2: Bad magician! Case #3: Volunteer cheated!

```
2
1 2 3 4
5 6 7 8
9 10 11 12
13 14 15 16
2
1 2 3 4
5 6 7 8
9 10 11 12
13 14 15 16
2
1 2 3 4
5 6 7 8
9 10 11 12
13 14 15 16
3
1 2 3 4
5 6 7 8
9 10 11 12
13 14 15 16
```

Problem B. Cookie Clicker Alpha

Small input 8 points	Solve B-small
Large input 11 points	Solve B-large

Introduction

Cookie Clicker is a Javascript game by Orteil, where players click on a picture of a giant cookie. Clicking on the giant cookie gives them cookies. They can spend those cookies to buy buildings. Those buildings help them get even more cookies. Like this problem, the game is very cookie-focused. This problem has a similar idea, but it does not assume you have played *Cookie Clicker*. Please don't go play it now: it might be a long time before you come back.

Problem

In this problem, you start with 0 cookies. You gain cookies at a rate of 2 cookies per second, by clicking on a giant cookie. Any time you have at least **C** cookies, you can buy a cookie farm. Every time you buy a cookie farm, it costs you **C** cookies and gives you an extra **F** cookies per second.

Once you have **X** cookies that you haven't spent on farms, you win! Figure out how long it will take you to win if you use the best possible strategy.

Example

Suppose **C**=500.0, **F**=4.0 and **X**=2000.0. Here's how the best possible strategy plays out:

1. You start with 0 cookies, but producing 2 cookies per second.
2. After **250** seconds, you will have **C**=500 cookies and can buy a farm that produces **F**=4 cookies per second.
3. After buying the farm, you have 0 cookies, and your total cookie production is 6 cookies per second.
4. The next farm will cost 500 cookies, which you can buy after about **83.3333333** seconds.
5. After buying your second farm, you have 0 cookies, and your total cookie production is 10 cookies per second.
6. Another farm will cost 500 cookies, which you can buy after **50** seconds.
7. After buying your third farm, you have 0 cookies, and your total cookie production is 14 cookies per second.
8. Another farm would cost 500 cookies, but it actually makes sense not to buy it: instead you can just wait until you have **X**=2000 cookies, which takes about **142.8571429** seconds.

Total time: 250 + 83.3333333 + 50 + 142.8571429 = 526.1904762 seconds.

Notice that you get cookies continuously: so 0.1 seconds after the game starts you'll have 0.2 cookies, and π seconds after the game starts you'll have 2π cookies.

Input

The first line of the input gives the number of test cases, **T**. **T** lines follow. Each line contains three space-separated real-valued numbers: **C**, **F** and **X**, whose meanings are described earlier in the problem statement.

C, **F** and **X** will each consist of at least 1 digit followed by 1 decimal point followed by from 1 to 5 digits. There will be no leading zeroes.

Output

For each test case, output one line containing "Case #x: y", where x is the test case number (starting from 1) and y is the minimum number of seconds it takes before you can have **X** delicious cookies.

We recommend outputting y to 7 decimal places, but it is not required. y will be considered correct if it is close enough to the correct number: within an absolute or relative error of 10^{-6}. See the FAQ for an explanation of what that means, and what formats of real numbers we accept.

Limits

$1 \leq \mathbf{T} \leq 100$.

Small dataset

$1 \le \mathbf{C} \le 500$.
$1 \le \mathbf{F} \le 4$.
$1 \le \mathbf{X} \le 2000$.

Large dataset

$1 \le \mathbf{C} \le 10000$.
$1 \le \mathbf{F} \le 100$.
$1 \le \mathbf{X} \le 100000$.

Sample

Input	Output
4 30.0 1.0 2.0 30.0 2.0 100.0 30.50000 3.14159 1999.19990 500.0 4.0 2000.0	Case #1: 1.0000000 Case #2: 39.1666667 Case #3: 63.9680013 Case #4: 526.1904762

Note

Cookie Clicker was created by Orteil. Orteil does not endorse and has no involvement with Google Code Jam.

Problem C. Minesweeper Master

Small input 11 points	Solve C-small
Large input 24 points	Solve C-large

Problem

Minesweeper is a computer game that became popular in the 1980s, and is still included in some versions of the *Microsoft Windows* operating system. This problem has a similar idea, but it does not assume you have played *Minesweeper*.

In this problem, you are playing a game on a grid of identical cells. The content of each cell is initially hidden. There are **M** mines hidden in **M** different cells of the grid. No other cells contain mines. You may click on any cell to reveal it. If the revealed cell contains a mine, then the game is over, and you lose. Otherwise, the revealed cell will contain a digit between 0 and 8, inclusive, which corresponds to the number of neighboring cells that contain mines. Two cells are neighbors if they share a corner or an edge. Additionally, if the revealed cell contains a 0, then all of the neighbors of the revealed cell are automatically revealed as well, recursively. When all the cells that don't contain mines have been revealed, the game ends, and you win.

For example, an initial configuration of the board may look like this ('*' denotes a mine, and 'c' is the first clicked cell):

```
*..*...**.
....*.....
..c..*....
........*.
..........
```

There are no mines adjacent to the clicked cell, so when it is revealed, it becomes a 0, and its 8 adjacent cells are revealed as well. This process continues, resulting in the following board:

```
*..*...**.
1112*.....
00012*....
00001111*.
00000001..
```

At this point, there are still un-revealed cells that do not contain mines (denoted by '.' characters), so the player has to click again in order to continue the game.

You want to win the game as quickly as possible. There is nothing quicker than winning in one click. Given the size of the board (**R** x **C**) and the number of hidden mines **M**, is it possible (however unlikely) to win in one click? You may choose where you click. If it is possible, then print any valid mine configuration and the coordinates of your click, following the specifications in the *Output* section. Otherwise, print "Impossible".

Input

The first line of the input gives the number of test cases, **T**. **T** lines follow. Each line contains three space-separated integers: **R**, **C**, and **M**.

Output

For each test case, output a line containing "Case #x:", where x is the test case number (starting from 1). On the following **R** lines, output the board configuration with **C** characters per line, using '.' to represent an empty cell, '*' to represent a cell that contains a mine, and 'c' to represent the clicked cell.

If there is no possible configuration, then instead of the grid, output a line with "Impossible" instead. If there are multiple possible configurations, output any one of them.

Limits

$0 \leq \mathbf{M} < \mathbf{R} * \mathbf{C}$.

Small dataset

$1 \le \mathbf{T} \le 230$.
$1 \le \mathbf{R}, \mathbf{C} \le 5$.

Large dataset

$1 \le \mathbf{T} \le 140$.
$1 \le \mathbf{R}, \mathbf{C} \le 50$.

Sample

Input	Output
5 5 5 23 3 1 1 2 2 1 4 7 3 10 10 82	Case #1: Impossible Case #2: c . * Case #3: Impossible Case #4: .....* .c....* ....*.... Case #5: ********** ********** ********** ****....** ***.....** ***.c...** ***....*** ********** ********** **********

Problem D. Deceitful War

This problem is the hardest problem to understand in this round. If you are new to Code Jam, you should probably try to solve the other problems first.

Problem

Naomi and Ken sometimes play games together. Before they play, each of them gets **N** identical-looking blocks of wood with masses between 0.0kg and 1.0kg (exclusive). All of the blocks have different weights. There are lots of games they could play with those blocks, but they usually play something they call War. Here is how War works:

1. Each player weighs each of his or her own blocks, so each player knows the weights of all of his or her own blocks, but not the weights of the other player's blocks.
2. They repeat the following process **N** times:
    1. Naomi chooses one of her own blocks, with mass $\texttt{Chosen}_{\texttt{Naomi}}$.
    2. Naomi tells Ken the mass of the block she chose.
    3. Ken chooses one of his own blocks, with mass $\texttt{Chosen}_{\texttt{Ken}}$.
    4. They each put their block on one side of a balance scale, and the person whose block is heavier gets one point.
    5. Both blocks are destroyed in a fire.

Naomi has realized three things about War. First, she has realized that she loses a lot. Second, she has realized that there is a unique strategy that Ken can follow to maximize his points without assuming anything about Naomi's strategy, and that Ken always uses it. Third, she has realized that she hates to lose. Naomi has decided that instead of playing War, she will play a game she calls Deceitful War. The great thing about Deceitful War is that Ken will think they're playing War!

Here is how Deceitful War works, with differences between Deceitful War and War in bold:

1. Each player weighs each of his or her own blocks. **Naomi also weighs Ken's blocks while he isn't looking, so Naomi knows the weights of all blocks** and Ken only knows the weights of his own blocks.
2. They repeat the following process **N** times:
    1. Naomi chooses one of her own blocks, with mass $\texttt{Chosen}_{\texttt{Naomi}}$.
    2. Naomi tells Ken **a number, $\texttt{Told}_{\texttt{Naomi}}$, between 0.0kg and 1.0kg exclusive.** Ken, who thinks they're playing War, thinks the number Naomi just told him is $\texttt{Chosen}_{\texttt{Naomi}}$.
    3. Ken chooses one of his own blocks, with mass $\texttt{Chosen}_{\texttt{Ken}}$.
    4. They each put their block on one side of a balance scale, and the person whose block is heavier gets one point.
    5. Both blocks are destroyed in a fire.

Naomi doesn't want Ken to know that she isn't playing War; so when she is choosing which block to play, and what mass to tell Ken, she must make sure that the balance scale won't reveal that $\texttt{Chosen}_{\texttt{Naomi}} \neq \texttt{Told}_{\texttt{Naomi}}$. In other words, she must make decisions so that:

- $\texttt{Chosen}_{\texttt{Naomi}} > \texttt{Chosen}_{\texttt{Ken}}$ if, and only if, $\texttt{Told}_{\texttt{Naomi}} > \texttt{Chosen}_{\texttt{Ken}}$, and
- $\texttt{Told}_{\texttt{Naomi}}$ is not equal to the mass of any of Ken's blocks, because he knows that isn't possible.

It might seem like Naomi won't win any extra points by being deceitful, because Ken might discover that she wasn't playing War; but Naomi knows Ken thinks both players are playing War, and she knows what he knows, and she knows Ken will always follow his unique optimal strategy for War, so she can always predict what he will play.

You'll be given the masses of the blocks Naomi and Ken started with. Naomi will play Deceitful War optimally to gain the maximum number of points. Ken will play War optimally to gain the maximum number of points *assuming that both players are playing War*. What will Naomi's score be? What would it have been if she had played War optimally instead?

Examples

If each player has a single block left, where Naomi has 0.5kg and Ken has 0.6kg, then Ken is guaranteed to score the point. Naomi can't say her number is ≥ 0.6kg, or Ken will know she isn't playing War when the balance scale shows his block was heavier.

If each player has two blocks left, where Naomi has [0.7kg, 0.2kg] and Ken has [0.8kg, 0.3kg], then Naomi could choose her 0.2kg block, and deceive Ken by telling him that she chose a block that was 0.6kg. Ken assumes Naomi is telling the truth (as in how the War game works) and will play his 0.8kg block to score a point. Ken was just deceived, but he will never realize it because the balance scale shows that his 0.8kg block is, like he expected, heavier than the block Naomi played. Now Naomi can play her 0.7kg block, tell Ken it is 0.7kg, and score a point. If Naomi had played War instead of Deceitful War, then Ken would have scored two points and Naomi would have scored zero.

Input

The first line of the input gives the number of test cases, **T**. **T** test cases follow. Each test case starts with a line containing a single integer **N**, the number of blocks each player has. Next follows a line containing **N** space-separated real numbers: the masses of Naomi's blocks, in kg. Finally there will be a line containing **N** space-separated real numbers: the masses of Ken's blocks, in kg.

Each of the masses given to Ken and Naomi will be represented as a 0, followed by a decimal point, followed by 1-5 digits. Even though all the numbers in the input have 1-5 digits after the decimal point, Ken and Naomi don't know that; so Naomi can still tell Ken that she played a block with mass 0.5000001kg, and Ken has no reason not to believe her.

Output

For each test case, output one line containing "Case #**x**: **y** **z**", where **x** is the test case number (starting from 1), **y** is the number of points Naomi will score if she plays Deceitful War optimally, and **z** is the number of points Naomi will score if she plays War optimally.

Limits

1 ≤ **T** ≤ 50.
All the masses given to Ken and Naomi are distinct, and between 0.0 and 1.0 exclusive.

Small dataset

1 ≤ **N** ≤ 10.

Large dataset

1 ≤ **N** ≤ 1000.

Sample

Input

```
4
1
0.5
0.6
2
0.7 0.2
0.8 0.3
3
0.5 0.1 0.9
0.6 0.4 0.3
9
0.186 0.389 0.907 0.832 0.959 0.557 0.300 0.992 0.899
0.916 0.728 0.271 0.520 0.700 0.521 0.215 0.341 0.458
```

Output

```
Case #1: 0 0
Case #2: 1 0
Case #3: 2 1
Case #4: 8 4
```

Powered by

Google Cloud Platform

Round 1A 2014

A. Charging Chaos
B. Full Binary Tree
C. Proper Shuffle

Contest Analysis
Questions asked

Problem A. Charging Chaos

Small input 8 points	Solve A-small
Large input 17 points	Solve A-large

Problem

Shota the farmer has a problem. He has just moved into his newly built farmhouse, but it turns out that the outlets haven't been configured correctly for all of his devices. Being a modern farmer, Shota owns a large number of smartphones and laptops, and even owns a tablet for his favorite cow Wagyu to use. In total, he owns **N** different devices.

As these devices have different specifications and are made by a variety of companies, they each require a different electric flow to charge. Similarly, each outlet in the house outputs a specific electric flow. An electric flow can be represented by a string of 0s and 1s of length **L**.

Shota would like to be able to charge all **N** of his devices at the same time. Coincidentally, there are exactly **N** outlets in his new house. In order to configure the electric flow from the outlets, there is a master control panel with **L** switches. The i^{th} switch flips the i^{th} bit of the electric flow from each outlet in the house. For example, if the electric flow from the outlets is:

```
Outlet 0: 10
Outlet 1: 01
Outlet 2: 11
```

Then flipping the second switch will reconfigure the electric flow to:

```
Outlet 0: 11
Outlet 1: 00
Outlet 2: 10
```

If Shota has a smartphone that needs flow "11" to charge, a tablet that needs flow "10" to charge, and a laptop that needs flow "00" to charge, then flipping the second switch will make him very happy!

Misaki has been hired by Shota to help him solve this problem. She has measured the electric flows from the outlets in the house, and noticed that they are all different. Decide if it is possible for Shota to charge all of his devices at the same time, and if it is possible, figure out the minimum number of switches that needs to be flipped, because the switches are big and heavy and Misaki doesn't want to flip more than what she needs to.

Input

The first line of the input gives the number of test cases, **T**. **T** test cases follow. Each test case consists of three lines. The first line contains two space-separated integers **N** and **L**. The second line contains **N** space-separated strings of length **L**, representing the initial electric flow from the outlets. The third line also contains **N** space-separated strings of length **L**, representing the electric flow required by Shota's devices.

Output

For each test case, output one line containing "Case #**x**: **y**", where **x** is the case number (starting from 1) and **y** is the minimum number of switches to be flipped in order for Shota to charge all his devices. If it is impossible, **y** should be the string "NOT POSSIBLE" (without the quotes). Please note that our judge is not case-sensitive, but it is strict in other ways: so although "not possible" will be judged correct, any misspelling will be judged wrong. We suggest copying/pasting the string NOT POSSIBLE into your code.

Limits

$1 \leq \mathbf{T} \leq 100$.
No two outlets will be producing the same electric flow, initially.
No two devices will require the same electric flow.

Small dataset

$1 \leq \mathbf{N} \leq 10$.
$2 \leq \mathbf{L} \leq 10$.

Large dataset

$1 \leq \mathbf{N} \leq 150$.
$10 \leq \mathbf{L} \leq 40$.

Sample

Input	Output
3 3 2 01 11 10 11 00 10 2 3 101 111 010 001 2 2 01 10 10 01	Case #1: 1 Case #2: NOT POSSIBLE Case #3: 0

Explanation

In the first example case, Misaki can flip the second switch once. The electric flow from the outlets becomes:

```
Outlet 0: 00
Outlet 1: 10
Outlet 2: 11
```

Then Shota can use the outlet 0 to charge device 1, the outlet 1 to charge device 2, outlet 2 to charge device 0. This is also a solution that requires the minimum amount number of switches to be flipped.

Round 1A 2014

A. Charging Chaos
B. Full Binary Tree
C. Proper Shuffle

Contest Analysis
Questions asked

Problem B. Full Binary Tree

Small input 9 points	Solve B-small
Large input 21 points	Solve B-large

Problem

A tree is a connected graph with no cycles.

A rooted tree is a tree in which one special vertex is called the root. If there is an edge between **X** and **Y** in a rooted tree, we say that **Y** is a child of **X** if **X** is closer to the root than **Y** (in other words, the shortest path from the root to **X** is shorter than the shortest path from the root to **Y**).

A full binary tree is a rooted tree where every node has either exactly 2 children or 0 children.

You are given a tree **G** with **N** nodes (numbered from **1** to **N**). You are allowed to delete some of the nodes. When a node is deleted, the edges connected to the deleted node are also deleted. Your task is to delete as few nodes as possible so that the remaining nodes form a full binary tree for some choice of the root from the remaining nodes.

Input

The first line of the input gives the number of test cases, **T**. **T** test cases follow. The first line of each test case contains a single integer **N**, the number of nodes in the tree. The following **N**-1 lines each one will contain two space-separated integers: $\mathbf{X_i}$ $\mathbf{Y_i}$, indicating that **G** contains an undirected edge between $\mathbf{X_i}$ and $\mathbf{Y_i}$.

Output

For each test case, output one line containing "Case #**x**: **y**", where **x** is the test case number (starting from 1) and **y** is the minimum number of nodes to delete from **G** to make a full binary tree.

Limits

$1 \le \mathbf{T} \le 100$.
$1 \le \mathbf{X_i}, \mathbf{Y_i} \le \mathbf{N}$
Each test case will form a valid connected tree.

Small dataset

$2 \le \mathbf{N} \le 15$.

Large dataset

$2 \le \mathbf{N} \le 1000$.

Sample

Input	Output
3 3 2 1 1 3 7 4 5 4 2 1 2 3 1 6 4 3 7 4 1 2 2 3 3 4	Case #1: 0 Case #2: 2 Case #3: 1

In the first case, **G** is already a full binary tree (if we consider node 1 as the root), so we don't need to do anything.

In the second case, we may delete nodes 3 and 7; then 2 can be the root of a full binary tree.

In the third case, we may delete node 1; then 3 will become the root of a full binary tree (we could also have deleted node 4; then we could have made 2 the root).

Problem C. Proper Shuffle

Small input
45 points

Solve C-small

Problem

A *permutation* of size **N** is a sequence of **N** numbers, each between 0 and **N**-1, where each number appears exactly once. They may appear in any order.

There are many (**N** *factorial*, to be precise, but it doesn't matter in this problem) permutations of size **N**. Sometimes we just want to pick one at random, and of course we want to pick one at random *uniformly*: each permutation of size **N** should have the same probability of being chosen.

Here's the pseudocode for one of the possible algorithms to achieve that goal (we'll call it the *good* algorithm below):

```
for k in 0 .. N-1:
  a[k] = k
for k in 0 .. N-1:
  p = randint(k .. N-1)
  swap(a[k], a[p])
```

In the above code, `randint(a .. b)` returns a uniform random integer between **a** and **b**, inclusive.

Here's the same algorithm in words. We start with the *identity* permutation: all numbers from 0 to **N**-1 written in increasing order. Then, for each **k** between 0 and **N**-1, inclusive, we pick an independent uniform random integer $\mathbf{p_k}$ between **k** and **N**-1, inclusive, and swap the element at position **k** (0-based) in our permutation with the element at position $\mathbf{p_k}$.

Here's an example for **N**=4. We start with the identity permutation:

0 1 2 3

Now **k**=0, and we pick a random $\mathbf{p_0}$ between 0 and 3, inclusive. Let's say we picked 2. We swap the 0th and 2nd elements, and our permutation becomes:

2 1 0 3

Now **k**=1, and we pick a random $\mathbf{p_1}$ between 1 and 3, inclusive. Let's say we picked 2 again. We swap the 1st and 2nd elements, and our permutation becomes:

2 0 1 3

Now **k**=2, and we pick a random $\mathbf{p_2}$ between 2 and 3, inclusive. Let's say we picked 3. We swap the 2nd and 3rd elements, and our permutation becomes:

2 0 3 1

Now **k**=3, and we pick a random $\mathbf{p_3}$ between 3 and 3, inclusive. The only choice is 3. We swap the 3rd and 3rd elements, which means that the permutation doesn't change:

2 0 3 1

The process ends now, and this is our random permutation.

There are many other algorithms that produce a random permutation uniformly. However, there are also many algorithms to generate a random permutation that look very similar to this algorithm, but are not uniform — some permutations are more likely to be produced by those algorithms than others.

Here's one bad algorithm of this type. Take the *good* algorithm above, but at each step, instead of picking $\mathbf{p_k}$ randomly between **k** and **N**-1, inclusive, let's pick it randomly between 0 and **N**-1, inclusive. This is such a small change, but now some permutations are more likely to appear than others!

Here's the pseudocode for this algorithm (we'll call it the *bad* algorithm below):

```
for k in 0 .. N-1:
  a[k] = k
for k in 0 .. N-1:
  p = randint(0 .. N-1)
  swap(a[k], a[p])
```

In each test case, you will be given a permutation that was generated in the following way: first, we choose either the good or the bad algorithm described above, each with probability 50%. Then, we generate a permutation using the chosen algorithm. Can you guess which algorithm was chosen just by looking at the permutation?

Solving this problem

This problem is a bit unusual for Code Jam. You will be given **T** = 120 permutations of **N** = 1000 numbers each, and should print an answer for each permutation – this part is as usual. However, you don't need to get all of the answers correct! Your solution will be considered correct if your answers for at least **G** = 109 cases are correct. However, you must follow the output format, even for cases in which your answer doesn't turn out to be correct. The *only* thing that can be wrong on any case, yet still allow you to be judged correct, is swapping GOOD for BAD or vice versa; but you should still print either GOOD or BAD for each case.

It is guaranteed that the permutations given to you were generated according to the method above, and that they were generated independently of each other.

This problem involves randomness, and thus it might happen that even the best possible solution doesn't make 109 correct guesses for a certain input, as both the good and the bad algorithms can generate any permutation. Because of that, this problem doesn't have a Large input, and has just the Small input which you can try again if you think you got unlucky. Note that there is the usual 4-minute penalty for incorrect submissions if you later solve that input, even if the only reason you got it wrong was chance.

In our experience with this problem, that *did happen* (getting wrong answer just because of chance); so if you are confident that your solution should be working, but it failed, it might be a reasonable strategy to try again with the same solution which failed.

Good luck!

Input

The first line of the input gives the number of test cases, **T** (which will always be 120). Each test case contains two lines: the first line contains the single integer **N** (which will always be 1000), and the next line contains **N** space-separated integers - the permutation that was generated using one of the two algorithms.

Output

For each test case, output one line containing "Case #**x**: **y**", where **x** is the test case number (starting from 1) and **y** is either "GOOD" or "BAD" (without the quotes). You should output "GOOD" if you guess that the permutation was generated by the first algorithm described in the problem statement, and "BAD" if you guess that the permutation was generated by the second algorithm described in the problem statement.

Limits

T = 120
G = 109
N = 1000
Each number in the permutation will be between 0 and **N**-1 (inclusive), and each number from 0 to **N**-1 will appear exactly once in the permutation.

Sample

Input	Output
2 3 0 1 2 3 2 0 1	Case #1: BAD Case #2: GOOD

Note

The sample input doesn't follow the limitations from the problem statement - the real input will be much bigger.


Powered by
Google Cloud Platform

Problem A. The Repeater

Small input 10 points	Solve A-small
Large input 13 points	Solve A-large

Problem

Fegla and Omar like to play games every day. But now they are bored of all games, and they would like to play a new game. So they decided to invent their own game called "The Repeater".

They invented a 2 player game. Fegla writes down **N** strings. Omar's task is to make all the strings identical, if possible, using the minimum number of actions (possibly 0 actions) of the following two types:

- Select any character in any of the strings and repeat it (add another instance of this character exactly after it). For example, in a single move Omar can change "abc" to "abbc" (by repeating the character 'b').
- Select any two adjacent and identical characters in any of the strings, and delete one of them. For example, in a single move Omar can change "abbc" to "abc" (delete one of the 'b' characters), but can't convert it to "bbc".

The 2 actions are independent; it's not necessary that an action of the first type should be followed by an action of the second type (or vice versa).

Help Omar to win this game by writing a program to find if it is possible to make the given strings identical, and to find the minimum number of moves if it is possible.

Input

The first line of the input gives the number of test cases, **T**. **T** test cases follow. Each test case starts with a line containing an integer **N** which is the number of strings. Followed by **N** lines, each line contains a non-empty string (each string will consist of lower case English characters only, from 'a' to 'z').

Output

For each test case, output one line containing "Case #x: y", where x is the test case number (starting from 1) and y is the minimum number of moves to make the strings identical. If there is no possible way to make all strings identical, print "Fegla Won" (quotes for clarity).

Limits

$1 \leq \mathbf{T} \leq 100$.
1 ≤ length of each string ≤ 100.

Small dataset

$\mathbf{N} = 2$.

Large dataset

$2 \leq \mathbf{N} \leq 100$.

Sample

Input	Output
5 2 mmaw maw 2 gcj cj 3 aaabbb ab aabb 2 abc abc 3 aabc	Case #1: 1 Case #2: Fegla Won Case #3: 4 Case #4: 0 Case #5: 3

```
abbc
abcc
```

Problem B. New Lottery Game

Small input 8 points	Solve B-small
Large input 24 points	Solve B-large

New Lottery Game

The Lottery is changing! The Lottery used to have a machine to generate a random winning number. But due to cheating problems, the Lottery has decided to add another machine. The new winning number will be the result of the bitwise-AND operation between the two random numbers generated by the two machines.

To find the bitwise-AND of X and Y, write them both in binary; then a bit in the result in binary has a 1 if the corresponding bits of X and Y were both 1, and a 0 otherwise. In most programming languages, the bitwise-AND of X and Y is written X&Y.

For example:
The old machine generates the number 7 = 0111.
The new machine generates the number 11 = 1011.
The winning number will be (7 AND 11) = (0111 AND 1011) = 0011 = 3.

With this measure, the Lottery expects to reduce the cases of fraudulent claims, but unfortunately an employee from the Lottery company has leaked the following information: the old machine will always generate a non-negative integer less than **A** and the new one will always generate a non-negative integer less than **B**.

Catalina wants to win this lottery and to give it a try she decided to buy all non-negative integers less than **K**.

Given **A**, **B** and **K**, Catalina would like to know in how many different ways the machines can generate a pair of numbers that will make her a winner.

Could you help her?

Input

The first line of the input gives the number of test cases, **T**. **T** lines follow, each line with three numbers **A B K**.

Output

For each test case, output one line containing "Case #x: y", where x is the test case number (starting from 1) and y is the number of possible pairs that the machines can generate to make Catalina a winner.

Limits

$1 \leq \mathbf{T} \leq 100$.

Small dataset

$1 \leq \mathbf{A} \leq 1000$.
$1 \leq \mathbf{B} \leq 1000$.
$1 \leq \mathbf{K} \leq 1000$.

Large dataset

$1 \leq \mathbf{A} \leq 10^9$.
$1 \leq \mathbf{B} \leq 10^9$.
$1 \leq \mathbf{K} \leq 10^9$.

Sample

Input	Output
5 3 4 2 4 5 2 7 8 5 45 56 35 103 143 88	Case #1: 10 Case #2: 16 Case #3: 52 Case #4: 2411 Case #5: 14377

In the first test case, these are the 10 possible pairs generated by the old and new machine respectively that will make her a winner: <0,0>, <0,1>, <0,2>, <0,3>, <1,0>, <1,1>, <1,2>, <1,3>, <2,0> and <2,1>. Notice that <0,1> is not the same as <1,0>. Also, although the pair <2, 2> could be generated by the machines it wouldn't make Catalina win since (2 AND 2) = 2 and she only bought the numbers 0 and 1.

Problem C. The Bored Traveling Salesman

Small input 15 points	Solve C-small
Large input 30 points	Solve C-large

Problem

Your boss is sending you out on an international sales trip. What joy!

You have **N** cities (numbered from 1 to **N**) to visit and can get to them using a set of bidirectional flights that go between the cities.

All of the cities must be visited at least once. To do this you can book any number of tickets, subject to the following conditions:

- Each ticket consists of 2 flights, one from a specific city **X** to another specific city **Y** (this is called the **outbound** flight), and the other one from city **Y** to city **X** (this is called the **return** flight).
- You must use the outbound flight before the corresponding return flight (you can use other flights in between).
- At most 1 outbound flight going to each city, although there is no limit on the return flights (multiple return flights can go to the same city).
- You must use all flights which belong to the tickets you booked.
- You can otherwise visit the cities in any order you like.
- You can start your trip from any city you choose. You may not take an outbound flight to your starting city.

Now you could try to minimize the total distance travelled, but you did that last time, so that would be boring. Instead you noticed that each city has a distinct 5 digit ZIP (postal) code. When you visit a city for the first time (this includes the city which you start from) you write down the zip code and concatenate these into one large number (concatenate them in the order which you visited each city for the first time). What is the smallest number you can achieve?

Input

The first line of the input gives the number of test cases, **T**. **T** test cases follow.

Each test case starts with a single line containing two integers: the number of cities **N** and the number of possible bidirectional flights **M**.

N lines then follow, with the i-th line containing the 5-digit zip code of the i-th city. No ZIP code will have leading zeros and all ZIP codes in each test case will be distinct.

M lines then follow, each containing two integers **i** and **j** ($1 \leq i < j \leq N$) indicating that a bidirectional flight exists between the **i**-th city and the **j**-th city. All flights will be distinct within each test case.

It is guaranteed that you can visit every city following the rules above.

Output

For each test case, output one line containing "Case #x: y", where x is the test case number (starting from 1) and y is the smallest number you can achieve by concatenating the ZIP codes along your trip.

Limits

$1 \leq T \leq 100$.
$0 \leq M \leq N * (N - 1) / 2$.

Small dataset

$1 \leq N \leq 8$.

Large dataset

$1 \leq N \leq 50$.

Sample

Input	Output
4 3 2	Case #1: 100002000010001 Case #2: 109532844450012296513664 2

```
10001   Case #3: 1095328444366422965150012
20000   Case #4: 10001100021000310004100051006
10000
1 2
2 3
5 4
36642
28444
50012
29651
10953
1 4
2 3
2 5
4 5
5 5
36642
28444
50012
29651
10953
1 2
1 4
2 3
2 5
4 5
6 6
10001
10002
10003
10004
10005
10006
1 2
1 6
2 3
2 4
3 5
4 5
```

Explanation

In the last sample test case, the following is the sequence of what you should do to achieve the smallest number:

1. Start from city 1, write 10001.
2. Outbound flight from 1 to 2, write 10002.
3. Outbound flight from 2 to 3, write 10003.
4. Return flight from 3 to 2.
5. Outbound flight from 2 to 4, write 10004.
6. Outbound flight from 4 to 5, write 10005.
7. Return flight from 5 to 4.
8. Return flight from 4 to 2.
9. Return flight from 2 to 1.
10. Outbound flight from 1 to 6, write 10006.
11. Return flight from 6 to 1.

Problem A. Part Elf

This contest is open for practice. You can try every problem as many times as you like, though we won't keep track of which problems you solve. Read the Quick-Start Guide to get started.

Small input 8 points	Solve A-small
Large input 12 points	Solve A-large

Problem

Vida says she's part Elf: that at least one of her ancestors was an Elf. But she doesn't know if it was a parent (1 generation ago), a grandparent (2 generations ago), or someone from even more generations ago. Help her out!

Being part Elf works the way you probably expect. People who are Elves, Humans and part-Elves are all created in the same way: two parents get together and have a baby. If one parent is `A/B` Elf, and the other parent is `C/D` Elf, then their baby will be `(A/B + C/D) / 2` Elf. For example, if someone who is `0/1` Elf and someone who is `1/2` Elf have a baby, that baby will be `1/4` Elf.

Vida is certain about one thing: 40 generations ago, she had 2^{40} different ancestors, and each one of them was `1/1` Elf or `0/1` Elf.

Vida says she's **P**/**Q** Elf. Tell her what is the minimum number of generations ago that there could have been a `1/1` Elf in her family. If it is not possible for her to be **P**/**Q** Elf, tell her that she must be wrong!

Input

The first line of the input gives the number of test cases, **T**. **T** lines follow. Each contains a fraction of the form **P**/**Q**, where **P** and **Q** are integers.

Output

For each test case, output one line containing "Case #x: y", where x is the test case number (starting from 1) and y is the minimum number of generations ago a 1/1 Elf in her family could have been if she is **P**/**Q** Elf. If it's impossible that Vida could be **P**/**Q** Elf, y should be the string "impossible" (without the quotes).

Limits

$1 \le \mathbf{T} \le 100$.

Small dataset

$1 \le \mathbf{P} < \mathbf{Q} \le 1000$.
P and **Q** have no common factors. That means **P**/**Q** is a fraction in *lowest terms*.

Large dataset

$1 \le \mathbf{P} < \mathbf{Q} \le 10^{12}$.
P and **Q** *may* have common factors. **P**/**Q** is *not* guaranteed to be a fraction in lowest terms.

Sample

Input	Output
5 1/2 3/4 1/4 2/23 123/31488	Case #1: 1 Case #2: 1 Case #3: 2 Case #4: impossible Case #5: 8

Note that the fifth sample case does not meet the limits for the Small input. Even if you don't solve it correctly, you might still have solved the Small input correctly.

Explanation of sample cases

In the first sample case, Vida could have a `1/1` Elf parent and a `0/1` Elf parent. That means she could have had a 1/1 Elf one generation ago, so the answer is

1.

In the second sample case, Vida could have had a 1/1 Elf parent and a 1/2 Elf parent. That means she could have had a 1/1 Elf one generation ago, so the answer is 1.

In the third sample case, Vida could have had a 0/1 Elf parent and a 1/2 Elf parent. The 1/2 Elf parent could have had a 1/1 Elf parent and a 0/1 Elf parent. That means she could have had a 1/1 Elf two generations ago, so the answer is 2.

In the fourth sample case, it's impossible to be exactly 2/23 Elf if your ancestors 40 generations ago were all 0/1 Elf or 1/1 Elf.

Note

Yes, Vida has a lot of ancestors. If that is the part of the problem that seems the most unrealistic to you, please re-read the part about Elves.

Round 1C 2014

A. Part Elf
B. Reordering Train Cars
C. Enclosure

Contest Analysis
Questions asked 4

Problem B. Reordering Train Cars

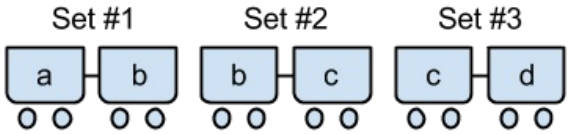
Small input 10 points	Solve B-small
Large input 25 points	Solve B-large

Problem

Yahya is a brilliant kid, so his mind raises a lot of interesting questions when he plays with his toys. Today's problem came about when his father brought him a set of train cars, where each car has a lowercase letter written on one side of the car.

When he first saw the gift, he was happy and started playing with them, connecting cars together without any particular goal. But after a while he got bored (as usual) from playing without having any goal. So, he decided to define a new interesting problem.

The problem is that he currently has **N** sets of connected cars. He can represent each set of connected cars as a string of lowercase letters. He wants to count the number of ways he can connect all **N** sets of cars to form one *valid* train. A train is valid if all occurrences of the same character are adjacent to each other.



The previous figure is one way Yahya could connect the cars "ab", "bc" and "cd" to make a valid train: "ab bc cd". If he had connected them in the order "cd ab bc", that would have been invalid: the "c" characters would not have been adjacent to each other.

You've surely noticed that this is not an easy problem for Yahya to solve, so he needs your help (and he is sure that you will give it!). That's it; go and help Yahya!

Note: letters are written only on one side of the cars, so you can not reverse them. For example, if a car has "ab" written on it, it could not be changed to read "ba".

Input

The first line of the input gives the number of test cases, **T**. **T** test cases follow. The first line of each test case contains a single integer **N**, the number of sets of connected cars. The following line contains **N** strings separated by a single space. Every given string represents a set of connected cars and is composed of lowercase English letters only.

Output

For each test case, output one line containing "Case #x: y", where x is the test case number (starting from 1) and y is the number of different ways of obtaining a valid train. As this number may be very big, output the number of ways *modulo 1,000,000,007*.

Limits

1 ≤ **T** ≤ 100.
1 ≤ Set of connected **Cars'** lengths ≤ 100.

Small dataset

1 ≤ **N** ≤ 10.

Large dataset

1 ≤ **N** ≤ 100.

Sample

Input	Output
3 3 ab bbbc cd 4 aa aa bc c 2 abc bcd	Case #1: 1 Case #2: 4 Case #3: 0

Sample Explanation

In the first case, there is only one way to form a valid train by joining string *"ab"* to *"bbbc"* to *"cd"* in this order.

While in the second case, there are 4 possible ways to form a valid train. Notice that there are two different sets of connected cars represented by the string *"aa"*, so there are two different ways to order these two strings and to group them to be one set of connected cars *"aaaa"*. Also there is only one way to order set of cars *"bc"* with *"c"* in only one way to be *"bcc"*. After that you can order *"aaaa"* and *"bcc"* in two different ways. So totally there are 2*2 = 4 ways to form a valid train.

In the third sample case, there is no possible way to form a valid train, as if joined in any of the two ways *"abc"+"bcd"* or *"bcd"+"abc"*, there will be two letters of *"b"* and *"c"* not consecutive.

Round 1C 2014

A. Part Elf

B. Reordering Train Cars

C. Enclosure

Contest Analysis

Questions asked 4

Submissions

Part Elf

8pt | Not attempted 4140/5606 users correct (74%)

12pt | Not attempted 2992/4086 users correct (73%)

Reordering Train Cars

10pt | Not attempted 1522/3094 users correct (49%)

25pt | Not attempted 516/847 users correct (61%)

Enclosure

15pt | Not attempted 521/1445 users correct (36%)

30pt | Not attempted 63/194 users correct (32%)

Top Scores

bmerry	100
Endagorion	100
yeputons	100
voover	100
Eryx	100
xiaowuc1	100
eurekash	100
stgatilov	100
Vasyl	100
Merkurev	100

Problem C. Enclosure

This contest is open for practice. You can try every problem as many times as you like, though we won't keep track of which problems you solve. Read the Quick-Start Guide to get started.

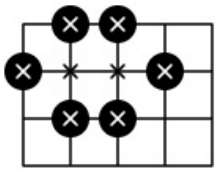
Small input 15 points	Solve C-small
Large input 30 points	Solve C-large

Problem

Your task in this problem is to find out the minimum number of stones needed to place on an N-by-M rectangular grid (N horizontal line segments and M vertical line segments) to enclose at least K intersection points. An intersection point is enclosed if either of the following conditions is true:

1. A stone is placed at the point.
2. Starting from the point, we cannot trace a path along grid lines to reach an empty point on the grid border through empty intersection points only.

For example, to enclose 8 points on a 4x5 grid, we need at least 6 stones. One of many valid stone layouts is shown below. Enclosed points are marked with an "x".



Input

The first line of the input gives the number of test cases, **T**. **T** lines follow. Each test case is a line of three integers: **N M K**.

Output

For each test case, output one line containing "Case #x: y", where x is the test case number (starting from 1) and y is the minimum number of stones needed.

Limits

$1 \le \mathbf{T} \le 100$.
$1 \le \mathbf{N}$.
$1 \le \mathbf{M}$.
$1 \le \mathbf{K} \le \mathbf{N} \times \mathbf{M}$.

Small dataset

$\mathbf{N} \times \mathbf{M} \le 20$.

Large dataset

$\mathbf{N} \times \mathbf{M} \le 1000$.

Sample

Input	Output
2 4 5 8 3 5 11	Case #1: 6 Case #2: 8

 Google Home - Terms and Conditions - Privacy Policies and Principles

Powered by

Google Cloud Platform

Problem A. Pegman

This contest is open for practice. You can try every problem as many times as you like, though we won't keep track of which problems you solve. Read the Quick-Start Guide to get started.

Small input 5 points	Solve A-small
Large input 10 points	Solve A-large

Problem

While using Google Street View, you may have picked up and dropped the character Pegman before. Today, a mischievous user is going to place Pegman in some cell of a rectangular grid of unit cells with **R** rows and **C** columns. Each of the cells in this grid might be blank, or it might be labeled with an arrow pointing in one of four possible directions: up, right, down, or left.

When Pegman is placed on a grid cell, if that cell is blank, Pegman stands still forever. However, if that cell has an arrow, Pegman starts to walk in that direction. As he walks, whenever he encounters a blank cell, he just keeps walking in his current direction, but whenever he encounters another arrow, he changes to the direction of that arrow and then keeps walking.

You know that it is possible that Pegman might keep happily walking around and around the grid forever, but it is also possible that Pegman's walk will take him over the edge of the grid! You may be able to prevent this and save him by changing the direction of one or more arrows. (Each arrow's direction can only be changed to one of the other three possible directions; arrows can only be changed, not added or removed.)

What is the smallest number of arrows you will need to change to ensure that Pegman will not walk off the edge, no matter where on the grid he is initially placed?

Input

The first line of the input gives the number of test cases, **T**. **T** test cases follow. Each begins with one line with two space-separated integers **R**, **C**. This line is followed by **R** lines, each of which has **C** characters, each of which describes a grid cell and is one of the following:

```
. period = no arrow
^ caret = up arrow
> greater than = right arrow
v lowercase v = down arrow
< less than = left arrow
```

Output

For each test case, output one line containing "Case #x: y", where x is the test case number (starting from 1) and y is the minimum number of arrows that must be changed to ensure that Pegman will not leave the grid no matter where he is initially placed, or the text `IMPOSSIBLE` if it is not possible to ensure this, no matter how many arrows you change.

Limits

1 ≤ **T** ≤ 100.

Small dataset

1 ≤ **R, C** ≤ 4.

Large dataset

1 ≤ **R, C** ≤ 100.

Sample

Input	Output
4 2 1 ^ ^ 2 2 >v ^< 3 3	Case #1: 1 Case #2: 0 Case #3: IMPOSSIBLE Case #4: 0

```
...
.^.
...
1 1
.
```

In Case #1, Pegman is guaranteed to walk off the top edge of the grid, no matter where he is placed. You can prevent that by changing the topmost arrow to point down, which will cause him to walk back and forth between those two arrows forever.

In Case #2, no matter where Pegman is placed, he will walk around and around the board clockwise in a circle. No arrows need to be changed.

In Case #3, the mischievous user might place Pegman on the up arrow in the middle of the grid, in which case he will start walking and then walk off the top edge of the grid. Changing the direction of this arrow won't help: it would just make him walk off a different edge.

In Case #4, the only possible starting cell is blank, so Pegman will stand still forever and is in no danger.

Problem B. Kiddie Pool

Small input
7 points
Solve B-small

Large input
18 points
Solve B-large

Problem

A kiddie pool is a big container in which you can put water, so that small children can play in it.

You have access to **N** different sources of water. The i^{th} source of water produces water at rate $\mathbf{R}_i$ and at temperature $\mathbf{C}_i$. Initially, all of the water sources are off. Each source of water can be switched on only once, and switched off only once; the act of switching a source on or off takes no additional time. Multiple sources can be on at the same time.

Your pool can hold an infinite amount of water, but you want to fill the pool to a volume of exactly **V** with a temperature of exactly **X**, as quickly as possible. If you turn sources on and off optimally (not every source necessarily has to be used), what's the minimum number of seconds it will take you to do this?

For the purposes of this problem, combining water that has volume $\mathbf{V}_0$ and temperature $\mathbf{X}_0$ with water that has volume $\mathbf{V}_1$ and temperature $\mathbf{X}_1$ will **instantaneously** create water with volume $\mathbf{V}_0+\mathbf{V}_1$ and temperature $(\mathbf{V}_0\mathbf{X}_0 + \mathbf{V}_1\mathbf{X}_1) / (\mathbf{V}_0 + \mathbf{V}_1)$. For example, combining 5L of water at 10 degrees with 10L of water at 40 degrees will result in 15L of water at 30 degrees. You should also assume that water does not heat or cool over time except as a result of being combined with other water.

Input

The first line of the input gives the number of test cases, **T**. **T** test cases follow. The first line of each test case contains three space-separated numbers: an integer **N**, and real numbers **V** and **X** as described above.

The next **N** lines each contain two space-separated real numbers, $\mathbf{R}_i$ and $\mathbf{C}_i$, the rate of flow and temperature of the i^{th} water source respectively. The volume is expressed in liters, rates of flow are expressed in liters per second, and temperatures are expressed in degrees Celsius.

All real numbers will be exactly specified to four decimal places.

Output

For each test case, output one line containing "Case #x: y", where x is the test case number (starting from 1) and y is the minimum number of seconds it takes to fill the kiddie pool to the right volume and temperature. If it is impossible to do so given the inputs, y should be the string `IMPOSSIBLE`.

y will be considered correct if it is within an absolute or relative error of 10^{-6} of the correct answer. See the FAQ for an explanation of what that means, and what formats of real numbers we accept. You can also read there about the format of our input real numbers, in which the decimal point will be represented by the `.` character.

Limits

$1 \le \mathbf{T} \le 100$.
$0.1 \le \mathbf{X} \le 99.9$.
$0.1 \le \mathbf{C}_i \le 99.9$.

Small dataset

$1 \le \mathbf{N} \le 2$.
$0.0001 \le \mathbf{V} \le 100.0$.
$0.0001 \le \mathbf{R}_i \le 100.0$.

Large dataset

$1 \le \mathbf{N} \le 100$.
$0.0001 \le \mathbf{V} \le 10000.0$.
$0.0001 \le \mathbf{R}_i \le 10000.0$.

Sample

Input	Output
6 1 10.0000 50.0000 0.2000 50.0000 2 30.0000 65.4321 0.0001 50.0000 100.0000 99.9000 2 5.0000 99.9000 30.0000 99.8999 20.0000 99.7000 2 0.0001 77.2831 0.0001 97.3911 0.0001 57.1751 2 100.0000 75.6127 70.0263 75.6127 27.0364 27.7990 4 5000.0000 75.0000 10.0000 30.0000 20.0000 50.0000 300.0000 95.0000 40.0000 2.0000	Case #1: 50.0000000 Case #2: 207221.843687375 Case #3: IMPOSSIBLE Case #4: 0.500000000 Case #5: 1.428034895 Case #6: 18.975332068

Note that Case #6 is not within the limits for the Small dataset.

In Case #1, the one available source happens to be the exact temperature we need. The optimal strategy is to immediately turn it on and let it run until we have 10 L. Since 0.2 L will come out every second, this takes 50 seconds.

In Case #2, one optimal strategy is to turn on the first source and let it run for 207221.843687375 seconds, and then, about 0.092778156 seconds before the end, also turn on the second source.

In Case #3, both available water sources are cooler than the target temperature, so there is no way to reach it.

Problem C. Bilingual

Small input 6 points	Solve C-small
Large input 24 points	Solve C-large

Problem

Elliot's parents speak French and English to him at home. He has heard a lot of words, but it isn't always clear to him which word comes from which language! Elliot knows one sentence that he's sure is English and one sentence that he's sure is French, and some other sentences that could be either English or French. If a word appears in an English sentence, it must be a word in English. If a word appears in a French sentence, it must be a word in French.

Considering all the sentences that Elliot has heard, what is the minimum possible number of words that he's heard that must be words in both English and French?

Input

The first line of the input gives the number of test cases, **T**. **T** test cases follow. Each starts with a single line containing an integer **N**. **N** lines follow, each of which contains a series of space-separated "words". Each "word" is made up only of lowercase characters a-z. The first of those **N** lines is a "sentence" in English, and the second is a "sentence" in French. The rest could be "sentences" in either English or French. (Note that the "words" and "sentences" are not guaranteed to be valid in any real language.)

Output

For each test case, output one line containing "Case #x: y", where x is the test case number (starting from 1) and y is the minimum number of words that Elliot has heard that must be words in both English and French.

Limits

$1 \leq \mathbf{T} \leq 25$.
Each word will contain no more than 10 characters.
The two "known" sentences will contain no more than 1000 words each.
The "unknown" sentences will contain no more than 10 words each.

Small dataset

$2 \leq \mathbf{N} \leq 20$.

Large dataset

$2 \leq \mathbf{N} \leq 200$.

Sample

Input	Output
4 2 he loves to eat baguettes il aime manger des baguettes 4 a b c d e f g h i j a b c i j f g h d e 4 he drove into a cul de sac elle a conduit sa voiture il a conduit dans un cul de sac il mange pendant que il conduit sa voiture 6 adieu joie de vivre je ne regrette rien adieu joie de vivre je ne regrette rien a b c d e f g h i j a b c i j f g h d e	Case #1: 1 Case #2: 4 Case #3: 3 Case #4: 8

In Case #1, Elliot knows for sure that the first sentence is in English and the second is in French, so there is no ambiguity; the only word that must be in both English and French is "baguettes".

In Case #2, the last two sentences could either be: English English, English French, French English, or French French. The second of those possibilities is the one that minimizes the number of words common to both languages; that set turns out to be d, e, i, and j.

Problem D. Drum Decorator

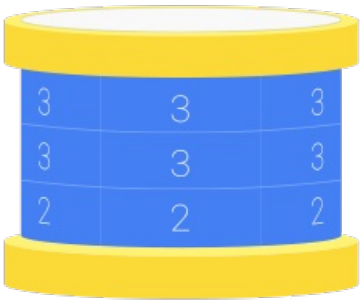
Small input 11 points	Solve D-small
Large input 19 points	Solve D-large

Problem

You are the drummer in the rock band Denise and the Integers. Your drum is a cylinder around which you've wrapped a rectangular grid of cells.

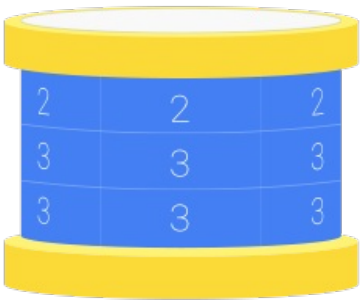
Your band is scheduled to perform in Mathland. The Mathlanders are a tough audience, and they will expect every cell of your drum to contain a positive integer; zeroes and negative integers are not allowed. Moreover, each integer K must border (share an edge, and not just a point, with) exactly K other cells with the same integer -- that is, a cell with a 1 must touch exactly one other cell with a 1, a cell with a 2 must touch exactly 2 other cells with a 2, and so on. Apart from this restriction, it does not matter what other cells a cell touches. (The circular top and bottom of the drum do not count as cells and do not need to be decorated. Note that this means that the cells along the top and bottom of the drum only touch three other cells each, whereas all the other cells touch four other cells each.)

For example, this is a valid decoration of a cylinder formed by a grid with 3 rows and 5 columns:



(Imagine that the unseen two columns on the back of the drum are the same as the three visible columns.)

You want to know how many different valid decorations are possible. Two decorations are different if one cannot be rotated (around the cylinder's axis of symmetry) to produce the other. The top and bottom of a drum are considered different, so this decoration of a 3x5 grid is different from the one above:



(Again, imagine that the unseen two columns on the back of the drum are the same as the three visible columns.)

Your drum has **R** rows and **C** columns. How many different valid decorations are possible? The number may be large, so return the number of decorations modulo $10^9 + 7$ (1000000007).

Input

The first line of the input gives the number of test cases, **T**. **T** lines follow; each contains two space-separated integers, **R** and **C**, which are the number of rows and columns in the drum.

Output

For each test case, output one line containing "Case #x: y", where x is the test case number (starting from 1) and y is the number of valid decorations modulo $10^9 + 7$, as described above.

Limits

Small dataset

$1 \leq \mathbf{T} \leq 20$.
$2 \leq \mathbf{R} \leq 6$.
$3 \leq \mathbf{C} \leq 6$.

Large dataset

$1 \leq \mathbf{T} \leq 100$.
$2 \leq \mathbf{R} \leq 100$.
$3 \leq \mathbf{C} \leq 100$.

Sample

Input	Output
2 2 4 3 5	Case #1: 1 Case #2: 2

In Case #1, the only solution is to fill all cells with 3s.

In Case #2, the only two solutions are the two depicted in the problem statement.

Round 3 2014

A. Magical, Marvelous Tour

B. Last Hit

C. Crime House

D. Willow

Contest Analysis

Questions asked

Problem A. Magical, Marvelous Tour

This contest is open for practice. You can try every problem as many times as you like, though we won't keep track of which problems you solve. Read the Quick-Start Guide to get started.

Small input 5 points	Solve A-small
Large input 8 points	Solve A-large

Problem

The mysterious owner of an electronics factory has decided to do something very intriguing. She has hidden *golden transistors* inside seven electronic devices, and the people who buy those devices will be invited to a magical, marvelous tour of the factory.

Arnar and Solveig have received a tip that there is a golden transistor hidden inside one device in their local electronics store. First they pooled their money together and bought all the devices, then placed them in a straight line, numbering the devices 0 to **N**-1. Each device has some number of transistors in it. Then they agreed on a strategy to decide who gets the golden transistor:

First, Arnar will select a range `[a, b]` (inclusive) of the devices, where `0 ≤ a ≤ b < N`. Next, Solveig will choose which one set of devices she wants to take:

- If a > 0, she may take all the devices in the range `[0, a-1]`.
- If b < **N**-1, she may take all the devices in the range `[b+1, N-1]`.
- She may always choose to take all the devices in the range `[a, b]`.

Once Solveig has chosen one of the sets of devices, Arnar takes all the devices she did not take.

For example, if there are 3 devices and Arnar selects the range `[1, 1]`, Solveig may choose to take the range `[0, 0]`, the range `[1, 1]` or the range `[2, 2]`. On the other hand, if Arnar selects the range `[1, 2]`, then Solveig may choose to take the range `[0, 0]` or the range `[1, 2]`.

Given how many transistors are in each device, and that Arnar and Solveig will each try to maximize their probability of getting the golden transistor (which is maximized by taking electronics with the maximum number of transistors), what is Arnar's probability of getting the golden transistor and thus winning the magical, marvelous tour?

Input

The first line of the input gives the number of test cases, **T**. **T** lines follow. Each line contains five numbers: **N**, **p**, **q**, **r** and **s**. This indicates that there are **N** devices, and the i^{th} device contains ((`i` * **p** + **q**) MOD **r** + **s**) transistors. Remember that the devices are numbered from 0 to **N**-1.

Output

For each test case, output one line containing "Case #x: y", where x is the test case number (starting from 1) and y is Arnar's probability of winning the magical, marvelous tour.

y will be considered correct if it is within an absolute or relative error of 10^{-9} of the correct answer. See the FAQ for an explanation of what that means, and what formats of real numbers we accept.

Limits

$1 \le \mathbf{T} \le 100$.
$1 \le \mathbf{p} \le 10^6$.
$1 \le \mathbf{q} \le 10^6$.
$1 \le \mathbf{r} \le 10^6$.
$1 \le \mathbf{s} \le 10^6$.

Small dataset

$1 \le \mathbf{N} \le 1000$.

Large dataset

$1 \le \mathbf{N} \le 10^6$.

Sample

Input

```
8
1 1 1 1 1
10 17 1 7 1
2 100 100 200 1
20 17 3 23 100
10 999999 999999 1000000 1000000
2 1 1 1 1
3 1 99 100 1
999999 1000000 999999 1000000 1000000
```

Output

```
Case #1: 0.0000000000
Case #2: 0.6111111111
Case #3: 0.0098039216
Case #4: 0.6471920290
Case #5: 0.6000006000
Case #6: 0.5000000000
Case #7: 0.0291262136
Case #8: 0.6666666667
```

Note that the last sample case does not meet the limits for the Small dataset. You could have a correct solution for the Small dataset that returns the wrong answer, or runs for a very long time, on the last sample case.

Explanation of Sample Cases

In the first sample case, there is one electronic device with one transistor. Arnar must select the range [0, 0], and Solveig must choose to take all the devices in the range [0, 0]. Arnar can't possibly win the magical, marvelous tour.

In the second sample case, there are ten electronic devices, with the following numbers of transistors: `[2, 5, 1, 4, 7, 3, 6, 2, 5, 1]`. Arnar will choose the range [4, 5], which contains the devices with 7 and 3 transistors. Solveig will choose the range [6, 9], which contains the devices with 6, 2, 5 and 1 transistors, leaving Arnar with the first six devices, and a probability of 22/36 of winning the tour.

In the third sample case, the devices have 101 and 1 transistors.

In the fourth sample case, the devices have the following numbers of transistors: `[103, 120, 114, 108, 102, 119, 113, 107, 101, 118, 112, 106, 100, 117, 111, 105, 122, 116, 110, 104]`.

In the fifth sample case, the devices have the following numbers of transistors: `[1999999, 1999998, 1999997, 1999996, 1999995, 1999994, 1999993, 1999992, 1999991, 1999990]`.

In the sixth sample case, the devices both have 1 transistor.

In the seventh sample case, the devices have the following numbers of transistors: `[100, 1, 2]`.

Problem B. Last Hit

Small input 10 points	Solve B-small
Large input 14 points	Solve B-large

Problem

Diana needs your help maximizing her gold while playing her favorite game. She is often faced with a scenario where she is standing close to her tower and is facing **N** monsters. When that happens, Diana and the tower take turns shooting the monsters, and she goes first. During her turn, Diana *may* choose a monster to shoot at (this means Diana may choose to skip a turn). During its turn, the tower shoots the monster closest to it. Diana and the tower can not shoot dead monsters.

If Diana shoots at a monster, its hit points are reduced by **P**. If the tower shoots at a monster, its hit points are reduced by **Q**. If a monster's hit points are reduced below 1, it is killed. The i^{th} monster starts with $\mathbf{H_i}$ hit points. Diana is awarded $\mathbf{G_i}$ gold if her shot kills the i^{th} monster, but none if the tower's shot kills it. What is the maximum amount of gold Diana can obtain?

Input

The first line of the input gives the number of test cases, **T**. **T** test cases follow. Each case begins with one line containing three space-separated integers representing **P**, **Q** and **N**. **N** lines then follow, with the i^{th} line containing two space-separated integers representing $\mathbf{H_i}$ and $\mathbf{G_i}$.

The monsters are given in the order of their distance from the tower. In other words, the tower will shoot at the i^{th} monster only if all monsters < i are dead.

Output

For each test case, output one line containing "Case #x: y", where x is the case number (starting from 1) and y is the maximum amount of gold that Diana can obtain.

Limits

$1 \le \mathbf{T} \le 100$
$20 \le \mathbf{P} \le 200$
$20 \le \mathbf{Q} \le 200$
$1 \le \mathbf{H_i} \le 200$
$0 \le \mathbf{G_i} \le 10^6$

Small dataset

$1 \le \mathbf{N} \le 4$

Large dataset

$1 \le \mathbf{N} \le 100$

Sample

Input	Output
2 20 40 3 100 100 20 100 60 100 20 60 3 80 100 80 200 120 300	Case #1: 300 Case #2: 500

In the second example, Diana should give up the first monster. During her first two turns she should soften up the third monster bringing it down to 80 hp, allowing her to easily get the last shot on the second and the third monsters.

Problem C. Crime House

Small input 12 points	Solve C-small
Large input 22 points	Solve C-large

Problem

While working for the police, you've identified a house where people go to commit crimes, called Crime House. One day, you set up a camera over the door of the house and record a video.

You don't know how many people were in Crime House at the start of the day, but you can see people enter and leave through the front door. Unfortunately, because the people entering and leaving Crime House are criminals, sometimes they wear masks; and you aren't quite sure if the front door is the only way in or out.

Sometimes you can guess who was wearing a mask. If criminal #5 entered the house, then someone wearing a mask left, then criminal #5 entered the house again, then either the person wearing the mask was criminal #5, or there is another way out of Crime House.

At the end of the day, when Crime House has closed its doors for the night, you watch your video. Because you're an optimist, you want to figure out if it's *possible* that there are no other entrances or exits from crime house; and if so, you want to figure out the *minimum* number of people who could be in Crime House at the end of the day.

Input

The first line of the input gives the number of test cases, **T**. **T** test cases follow. Each test case starts with a line containing a single integer **N**, the number of times people pass through the front door of Crime House in the day. Next follows **N** lines, each of which contains information about one person entering or leaving Crime House through the front door.

That information consists of a single character, E or L, followed by a space and then an integer **id**. If the first character is E, that indicates someone entered Crime House through the front door; if it's L, someone left through the front door. If **id** is greater than zero, the person with that identifier entered or left Crime House. If **id** is zero, then the person who entered or left Crime House was wearing a mask, and we don't know who he or she was.

Output

For each test case, output one line containing "Case #x: y", where x is the test case number (starting from 1). If it's possible that there are no other entrances or exits from Crime House, then y should be the minimum number of people who could be in Crime House at the end of the day. If that's impossible, y should be "CRIME TIME".

Limits

$1 \le \mathbf{T} \le 100$.
$0 \le \mathbf{id} \le 2000$.

Small dataset

$1 \le \mathbf{N} \le 15$.

Large dataset

$1 \le \mathbf{N} \le 1000$

Sample

Input	Output
5 3 E 5 L 0 E 5 2 L 1 L 1 4	Case #1: 1 Case #2: CRIME TIME Case #3: 1 Case #4: 4 Case #5: 0

```
L 1
E 0
E 0
L 1
7
L 2
E 0
E 1
E 2
E 0
E 3
L 4
13
L 4
L 1
L 2
E 0
L 1
E 0
L 2
E 0
L 2
E 0
E 0
L 1
L 4
```

Problem D. Willow

Problem

Hanaa and Sherine are playing Willow, a game that is played on a board containing **N** cities. The i^{th} city contains $\mathbf{C_i}$ coins, and there are **N** - 1 bidirectional roads running between the cities. All cities are reachable from one another. The game is played as follows:

First Hanaa chooses one of the cities as her starting location, then Sherine chooses one of the cities (possibly the same one Hanaa chose) as her starting location. Afterwards, they take turns playing the game, with Hanaa going first.

On a player's turn, that player *must* take all the coins on the city where she currently is, if there are any; there might be none if the city starts with no coins, or if one of the players has already started a turn in that city. Then, if possible, the player must travel to an adjacent city on a road. It might not be possible, because each road can be used at most once. This means that after one player has used a road, neither player is allowed to use the same road later. The game ends when neither Hanaa nor Sherine can make a move.

After the game ends, each player's score is equal to the difference between the number of coins she has and the number of coins her opponent has. If her opponent has more coins, this means that her score will be negative. Both players are trying to maximize their scores. Assuming that they are both using the best possible strategy to maximize their scores, what is the highest score that Hanaa can obtain?

Input

The first line of the input gives the number of test cases, **T**. **T** test cases follow. Each test case starts with a line containing one integer **N**, the number of cities on the board. **N** lines then follow, with the i^{th} line containing an integer $\mathbf{C_i}$, the number of coins in city i.

Finally there will be another **N** - 1 lines, with the i^{th} line (i starts from 1) containing a single integer j ($1 \leq i < j \leq \mathbf{N}$) indicating that there is a road between city i and city j. All cities are guaranteed to be reachable from one another at the start of the game.

Output

For each test case, output one line containing "Case #x: y", where x is the case number (starting from 1) and y is the highest score that Hanaa can obtain.

Limits

$1 \leq \mathbf{T} \leq 50$.
$0 \leq \mathbf{C_i} \leq 10000$.

Small dataset

$2 \leq \mathbf{N} \leq 80$.

Large dataset

For 10 test cases, $2 \leq \mathbf{N} \leq 4000$.
For the remaining test cases, $2 \leq \mathbf{N} \leq 500$.

Sample

Input	Output
3 3 1000 200 1000 2 3 8 8 0	Case #1: 200 Case #2: -2 Case #3: 5100

```
8
0
0
0
0
10
2
5
4
5
6
7
8
10
150
200
0
5000
0
100
0
0
0
10000
10
3
8
5
8
7
8
9
10
```

Submissions

Checkerboard Matrix

4pt | Not attempted **23/26 users** correct (88%)

9pt | Not attempted **23/23 users** correct (100%)

Power Swapper

4pt | Not attempted **25/25 users** correct (100%)

12pt | Not attempted **19/21 users** correct (90%)

Symmetric Trees

7pt | Not attempted **22/24 users** correct (92%)

18pt | Not attempted **15/22 users** correct (68%)

Paradox Sort

4pt | Not attempted **24/24 users** correct (100%)

28pt | Not attempted **11/15 users** correct (73%)

Allergy Testing

15pt | Not attempted **19/23 users** correct (83%)

35pt | Not attempted **1/6 users** correct (17%)

ARAM

22pt | Not attempted **3/5 users** correct (60%)

42pt | Not attempted **0/3 users** correct (0%)

Top Scores

Gennady.Korotkevich	136
eatmore	123
sevenkplus	101
mystic	95
mk.al13n	89
EgorKulikov	89
kcm1700	89
vepifanov	83
dzhulgakov	83
Romka	83

Problem A. Checkerboard Matrix

This contest is open for practice. You can try every problem as many times as you like, though we won't keep track of which problems you solve. Read the Quick-Start Guide to get started.

Small input 4 points	Solve A-small
Large input 9 points	Solve A-large

Problem

When she is bored, Mija sometimes likes to play a game with matrices. She tries to transform one matrix into another with the fewest moves. For Mija, one move is swapping any two rows of the matrix or any two columns of the matrix.

Today, Mija has a very special matrix **M**. **M** is a 2**N** by 2**N** matrix where every entry is either a 0 or a 1. Mija decides to try and transform **M** into a *checkerboard matrix* where the entries alternate between 0 and 1 along each row and column. Can you help Mija find the minimum number of moves to transform **M** into a *checkerboard matrix*?

Input

The first line of the input gives the number of test cases, **T**. **T** test cases follow. Each test case starts with a line containing a single integer: **N**. The next 2**N** lines each contain 2**N** characters which are the rows of **M**; each character is a 0 or 1.

Output

For each test case, output one line containing "Case #x: y", where x is the test case number (starting from 1) and y is the minimum number of row swaps and column swaps required to turn **M** into a *checkerboard matrix*. If it is impossible to turn **M** into a checkerboard matrix, y should be "IMPOSSIBLE".

Limits

$1 \leq \mathbf{T} \leq 100$.

Small dataset

$1 \leq \mathbf{N} \leq 10$.

Large dataset

$1 \leq \mathbf{N} \leq 10^3$.

Sample

Input	Output
3 1 01 10 2 1001 0110 0110 1001 1 00 00	Case #1: 0 Case #2: 2 Case #3: IMPOSSIBLE

In the first sample case, **M** is already a *checkerboard matrix*.

In the second sample case, Mija can turn **M** into a *checkerboard matrix* by swapping columns 1 and 2 and then swapping rows 1 and 2.

In the third sample case, Mija can never turn **M** into a *checkerboard matrix*; it doesn't have enough 1s.

Problem B. Power Swapper

Small input 4 points	Solve B-small
Large input 12 points	Solve B-large

Problem

In a parallel universe, people are crazy about using numbers that are powers of two, and they have defined an exciting sorting strategy for permutations of the numbers from 1 to $2^{\mathbf{N}}$. They have defined a swapping operation in the following way:

- A range of numbers to be swapped is valid if and only if it is a range of adjacent numbers of size 2^k, and its starting position (position of the first element in the range) is a multiple of 2^k (where positions are 0-indexed).
- A valid swap operation of *size-k* is defined by swapping two distinct, valid ranges of numbers, each of size 2^k.

To sort the given permutation, you are allowed to use at most one swap operation of each size k, for k in [0, **N**). Also, note that swapping a range with itself is not allowed.

For example, given the permutation [3, 6, 1, 2, 7, 8, 5, 4] (a permutation of the numbers from 1 to 2^3), the permutation can be sorted as follows:

- [3, 6, 1, 2, 7, 8, 5, 4]: make a *size-2* swap of the ranges [3, 6, 1, 2] and [7, 8, 5, 4].
- [7, 8, 5, 4, 3, 6, 1, 2]: make a *size-0* swap of [5] and [3].
- [7, 8, 3, 4, 5, 6, 1, 2]: make a *size-1* swap of [7, 8] and [1, 2].
- [1, 2, 3, 4, 5, 6, 7, 8]: done.

The previous steps used every swap size (0, 1, and 2) at most once. Also, notice that all the swaps were valid because both ranges for each size k started at indices that were multiples of 2^k.

Count how many ways there are to sort the given permutation by using the rules above. A way is an ordered sequence of swaps, and two ways are the same only if the sequences are identical.

Input

The first line of the input gives the number of test cases, **T**. **T** test cases follow. The first line of each test case contains a single integer **N**. The following line contains $\mathbf{2^N}$ space-separated integers: a permutation of the numbers 1, 2, ..., $2^{\mathbf{N}}$.

Output

For each test case, output one line containing "Case #x: y", where x is the test case number (starting from 1) and y is the number of ways to sort the given permutation using the rules above.

Limits

$1 \le \mathbf{T} \le 200$.

Small dataset

$1 \le \mathbf{N} \le 4$.

Large dataset

$1 \le \mathbf{N} \le 12$.

Sample

Input	Output
4 1 2 1 2 1 4 3 2 3 7 8 5 6 1 2 4 3	Case #1: 1 Case #2: 3 Case #3: 6 Case #4: 0

```
2
4 3 2 1
```

Problem C. Symmetric Trees

Small input 7 points	Solve C-small
Large input 18 points	Solve C-large

Problem

Given a vertex-colored tree with N nodes, can it be drawn in a 2D plane with a line of symmetry?

Formally, a tree is *line-symmetric* if each vertex can be assigned a location in the 2D plane such that:

- All locations are distinct.
- If vertex $\mathbf{v_i}$ has color **C** and coordinates ($\mathbf{x_i}$, $\mathbf{y_i}$), there must also be a vertex $\mathbf{v_i}'$ of color **C** located at (-$\mathbf{x_i}$, $\mathbf{y_i}$) -- Note if $\mathbf{x_i}$ is 0, $\mathbf{v_i}$ and $\mathbf{v_i}'$ are the same vertex.
- For each edge ($\mathbf{v_i}$, $\mathbf{v_j}$), there must also exist an edge ($\mathbf{v_i}'$, $\mathbf{v_j}'$).
- If edges were represented by straight lines between their end vertices, no two edges would share any points except where adjacent edges touch at their endpoints.

Input

The first line of the input gives the number of test cases, **T**. **T** test cases follow.

Each test case starts with a line containing a single integer **N**, the number of vertices in the tree.

N lines then follow, each containing a single uppercase letter. The i-th line represents the color of the i-th node.

N-1 lines then follow, each line containing two integers **i** and **j** ($1 \le \mathbf{i} < \mathbf{j} \le \mathbf{N}$). This denotes that the tree has an edge from the **i**-th vertex to the **j**-th vertex. The edges will describe a connected tree.

Output

For each test case, output one line containing "Case #x: y", where x is the case number (starting from 1) and y is "SYMMETRIC" if the tree is line-symmetric by the definition above or "NOT SYMMETRIC" if it isn't.

Limits

$1 \le \mathbf{T} \le 100$.

Small dataset

$2 \le \mathbf{N} \le 12$.

Large dataset

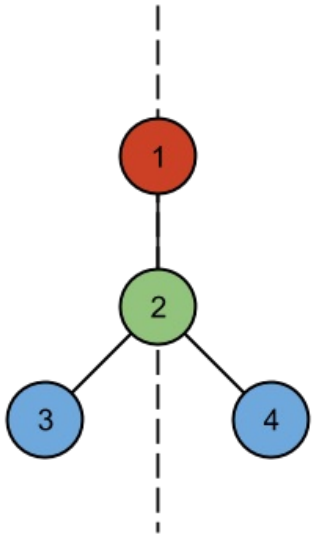
$2 \le \mathbf{N} \le 10000$.

Sample

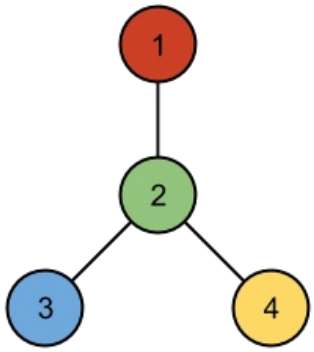
Input	Output
3 4 R G B B 1 2 2 3 2 4 4 R G B Y 1 2 2 3 2 4 12	Case #1: SYMMETRIC Case #2: NOT SYMMETRIC Case #3: SYMMETRIC

```
Y
B
Y
G
R
G
Y
Y
B
B
B
R
1 3
1 9
1 10
2 3
3 7
3 8
3 11
4 8
5 7
6 7
8 12
```

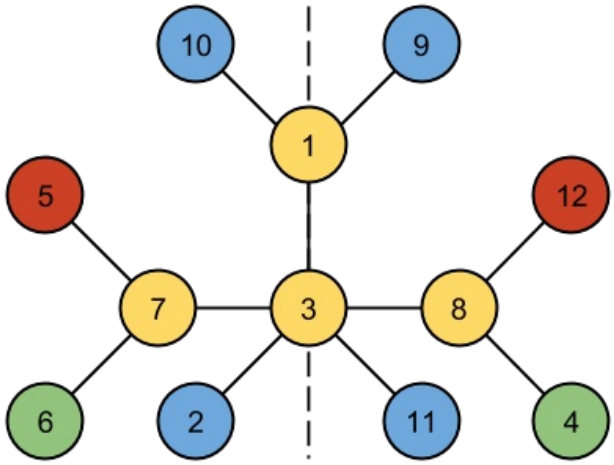
The first case can be drawn as follows:



No arrangement of the second case has a line of symmetry:



One way of drawing the third case with a symmetry line is as follows:



Problem D. Paradox Sort

Small input 4 points	Solve D-small
Large input 28 points	Solve D-large

Problem

Vlad likes candies. You have a bag of different candies, and you're going to let Vlad keep one of them. You choose an order for the candies, then give them to Vlad one at a time. For each candy Vlad receives (after the first one), he compares the candy he had to the one he was just given, keeps the one he likes more, and throws the other one away.

You would expect that for any order you choose, Vlad will always end up with his favorite candy. But this is not the case! He does not necessarily have a favorite candy. We know for any pair of candies which one he will prefer, but his choices do not necessarily correspond to a simple ranking. He may choose Orange when offered Orange and Lemon, Banana when offered Orange and Banana, and Lemon when offered Lemon and Banana!

There is a particular candy you want Vlad to end up with. Given Vlad's preferences for each pair of candies, determine if there is an ordering such that Vlad will end up with the right candy. If there is, find the lexicographically-smallest such ordering.

Input

The first line of the input gives the number of test cases, **T**. **T** test cases follow. Each test case will start with a line containing the integers **N** and **A**, separated by a space. **N** is the number of candies, and **A** is the number of the candy we want Vlad to finish with. The candies are numbered from 0 to **N**-1. The next **N** lines each contains **N** characters. Character j of line i will be 'Y' if Vlad prefers candy i to candy j, 'N' if Vlad prefers candy j to candy i, and '-' if i = j. Note that if i ≠ j, the jth character of the ith row must be different from the ith character of the jth row.

Output

For each test case output "Case #x: ", where x is the case number, followed by either "IMPOSSIBLE" or a space-separated list of the lexicographically-smallest ordering of candies that leaves Vlad with **A**.

Limits

1 ≤ **T** ≤ 100.

Small dataset

1 ≤ **N** ≤ 10.

Large dataset

1 ≤ **N** ≤ 100.

Sample

Input	Output
3 2 0 -Y N- 2 0 -N Y- 4 3 -YNN N-YY YN-Y YNN-	Case #1: 0 1 Case #2: IMPOSSIBLE Case #3: 1 2 0 3

Problem E. Allergy Testing

Small input 15 points	Solve E-small
Large input 35 points	Solve E-large

Problem

Kelly is allergic to exactly one of **N** foods, but she isn't sure which one. So she decides to undergo some experiments to find out.

In each experiment, Kelly picks several foods and eats them all. She waits **A** days to see if she gets any allergic reactions. If she doesn't, she knows she isn't allergic to any of the foods she ate. If she does get a reaction, she has to wait for it to go away: this takes a total of **B** days (measured from the moment when she ate the foods).

To simplify her experimentation, Kelly decides to wait until each experiment is finished (after **A** or **B** days) before starting the next one. At the start of each experiment, she can choose the set of foods she wants to eat based on the results of previous experiments.

Kelly chooses what foods to eat for each experiment to minimize the worst-case number of days before she knows which of the **N** foods she is allergic to. How long does it take her in the worst case?

Input

The first line of the input gives the number of test cases, **T**. **T** test cases follow. Each test case on a single line, containing three space-separated integers: **N**, **A** and **B**.

Output

For each test case, output one line containing "Case #x: y", where x is the test case number (starting from 1) and **y** is the number of days it will take for Kelly to find out which food she is allergic to, in the worst case.

Limits

$1 \le \mathbf{T} \le 200$.

Small dataset

$1 \le \mathbf{N} \le 10^{15}$.
$1 \le \mathbf{A} \le \mathbf{B} \le 100$.

Large dataset

$1 \le \mathbf{N} \le 10^{15}$.
$1 \le \mathbf{A} \le \mathbf{B} \le 10^{12}$.

Sample

Input	Output
3 4 5 7 8 1 1 1 23 32	Case #1: 12 Case #2: 3 Case #3: 0

In the first sample case:

- First, Kelly eats foods #1 and #2.
- If she gets no reaction after 5 days, she eats food #3. 5 days after *that*, she will know whether she is allergic to food #3 or food #4.
- If she does get a reaction to the first experiment, then 7 days after the first experiment, she eats food #1. 5 days after that, she will know whether she is allergic to food #1 or food #2.

Problem F. ARAM

Small input 22 points	Solve F-small
Large input 42 points	Solve F-large

Problem

In the game League of Legends™, you can play a type of game called "ARAM", which is short for "All Random, All Mid". This problem uses a similar idea, but doesn't require you to have played League of Legends to understand it.

Every time you start playing an ARAM game, you're assigned one of **N** "champions", uniformly at random. You're more likely to win with some champions than with others, so if you get unlucky then you might wish you'd been given a different champion. Luckily for you, the game includes a "Reroll" function.

Rerolling randomly reassigns you a champion in a way that will be described below; but you can't reroll whenever you want to. The ability to reroll works like a kind of money. Before you play your first ARAM game, you begin with **R** RD ("reroll dollars"). You can only reroll if you have at least 1 RD, and you must spend 1 RD to reroll. After every game, you gain 1/**G** RD (where **G** is an integer), but you can never have more than **R** RD: if you have **R** RD and then play a game, you'll still have **R** RD after that game.

If you have at least 1RD, and you choose to reroll, you will spend 1RD and be re-assigned one of the **N** champions, uniformly at random. There's some chance you might get the same champion you had at first. If you don't like the champion you rerolled, and you still have at least 1RD left, you can reroll again. As long as you have at least 1RD left, you can keep rerolling.

For example, if **R**=2 and **G**=2, and you use a reroll in your first game, then after your first game you will have 1.5 RD. If you play another game, this time without using a reroll, you will have 2.0 RD. If you play another game without using a reroll, you will still have 2.0 RD (because you can never have more than **R**=2). If you use two rerolls in your next game, then after that game you will have 0.5 RD.

You will be given the list of champions, and how likely you are to win a game if you play each of them. If you play 10^{100} games and choose your strategy optimally, what fraction of the games do you expect to win?

Input

The first line of the input gives the number of test cases, **T**. **T** test cases follow. Each starts with a line containing three space-separated integers: **N**, **R** and **G**. The next line contains **N** space-separated, real-valued numbers $\mathbf{P}_i$, indicating the probability that you will win if you play champion `i`.

Output

For each test case, output one line containing "Case #x: y", where x is the test case number (starting from 1) and y is the proportion of games you will win if you play 10^{100} games.

y will be considered correct if it is within an absolute or relative error of 10^{-10} of the correct answer. See the FAQ for an explanation of what that means, and what formats of real numbers we accept.

Limits

$1 \le \mathbf{T} \le 100$.
$0.0 \le \mathbf{P}_i \le 1.0$.
$\mathbf{P}_i$ will be expressed as a single digit, followed by a decimal point, followed by 4 digits.

Small dataset

$1 \le \mathbf{N} \le 1000$.
$1 \le \mathbf{R} \le 2$.
$1 \le \mathbf{G} \le 3$.

Large dataset

$1 \le \mathbf{N} \le 1000$.
$1 \le \mathbf{R} \le 20$.
$1 \le \mathbf{G} \le 20$.

Sample

Input

```
3
2 1 1
1.0000 0.0000
3 1 1
1.0000 0.0000 0.5000
6 2 3
0.9000 0.6000 0.5000 0.1000 0.2000 0.8000
```

Output

```
Case #1: 0.750000000000
Case #2: 0.666666666667
Case #3: 0.618728522337
```

Note

League of Legends is a trademark of Riot Games. Riot Games does not endorse and has no involvement with Google Code Jam.