Problem A. Tic-Tac-Toe-Tomek

Small input 10 points	Solve A-small
Large input 20 points	Solve A-large

Problem

Tic-Tac-Toe-Tomek is a game played on a 4 x 4 square board. The board starts empty, except that a single 'T' symbol may appear in one of the 16 squares. There are two players: X and O. They take turns to make moves, with X starting. In each move a player puts her symbol in one of the empty squares. Player X's symbol is 'X', and player O's symbol is 'O'.

After a player's move, if there is a row, column or a diagonal containing 4 of that player's symbols, or containing 3 of her symbols and the 'T' symbol, she wins and the game ends. Otherwise the game continues with the other player's move. If all of the fields are filled with symbols and nobody won, the game ends in a draw. See the sample input for examples of various winning positions.

Given a 4 x 4 board description containing 'X', 'O', 'T' and '.' characters (where '.' represents an empty square), describing the current state of a game, determine the status of the Tic-Tac-Toe-Tomek game going on. The statuses to choose from are:

- "X won" (the game is over, and X won)
- "O won" (the game is over, and O won)
- "Draw" (the game is over, and it ended in a draw)
- "Game has not completed" (the game is not over yet)

If there are empty cells, and the game is not over, you should output "Game has not completed", even if the outcome of the game is inevitable.

Input

The first line of the input gives the number of test cases, **T**. **T** test cases follow. Each test case consists of 4 lines with 4 characters each, with each character being 'X', 'O', '.' or 'T' (quotes for clarity only). Each test case is followed by an empty line.

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 the statuses given above. Make sure to get the statuses exactly right. When you run your code on the sample input, it should create the sample output exactly, including the "Case #1: ", the capital letter "O" rather than the number "0", and so on.

Limits

The game board provided will represent a valid state that was reached through play of the game Tic-Tac-Toe-Tomek as described above.

Small dataset

1 ≤ **T** ≤ 10.

Large dataset

1 ≤ **T** ≤ 1000.

Sample

Input	Output
6 XXXT .... OO.. .... X0XT XX00 0X0X XX00 X0X.	Case #1: X won Case #2: Draw Case #3: Game has not completed Case #4: O won Case #5: O won Case #6: O won

```
0X..
....
....

00XX
0XXX
0X.T
0..0

XXX0
..0.
.0..
T...

0XXX
X0..
..0.
...0
```

Note

Although your browser might not render an empty line after the last test case in the sample input, in a real input file there would be one.

Problem B. Lawnmower

Small input 10 points	Solve B-small
Large input 30 points	Solve B-large

Problem

Alice and Bob have a lawn in front of their house, shaped like an **N** metre by **M** metre rectangle. Each year, they try to cut the lawn in some interesting pattern. They used to do their cutting with shears, which was very time-consuming; but now they have a new automatic lawnmower with multiple settings, and they want to try it out.

The new lawnmower has a height setting - you can set it to any height **h** between 1 and 100 millimetres, and it will cut all the grass higher than **h** it encounters to height **h**. You run it by entering the lawn at any part of the edge of the lawn; then the lawnmower goes in a straight line, perpendicular to the edge of the lawn it entered, cutting grass in a swath 1m wide, until it exits the lawn on the other side. The lawnmower's height can be set only when it is not on the lawn.

Alice and Bob have a number of various patterns of grass that they could have on their lawn. For each of those, they want to know whether it's possible to cut the grass into this pattern with their new lawnmower. Each pattern is described by specifying the height of the grass on each 1m x 1m square of the lawn.

The grass is initially 100mm high on the whole lawn.

Input

The first line of the input gives the number of test cases, **T**. **T** test cases follow. Each test case begins with a line containing two integers: **N** and **M**. Next follow **N** lines, with the *i*th line containing **M** integers $\mathbf{a_{i,j}}$ each, the number $\mathbf{a_{i,j}}$ describing the desired height of the grass in the *j*th square of the *i*th 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 either the word "YES" if it's possible to get the x-th pattern using the lawnmower, or "NO", if it's impossible (quotes for clarity only).

Limits

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

Small dataset

$1 \le \mathbf{N}, \mathbf{M} \le 10$.
$1 \le \mathbf{a_{i,j}} \le 2$.

Large dataset

$1 \le \mathbf{N}, \mathbf{M} \le 100$.
$1 \le \mathbf{a_{i,j}} \le 100$.

Sample

Input	Output
3 3 3 2 1 2 1 1 1 2 1 2 5 5 2 2 2 2 2 2 1 1 1 2 2 1 2 1 2 2 1 1 1 2 2 2 2 2 2 1 3 1 2 1	Case #1: YES Case #2: NO Case #3: YES

Problem C. Fair and Square

Small input 10 points	Solve C-small
Large input 1 35 points	Solve C-large-1
Large input 2 55 points	Solve C-large-2

Problem

Little John likes palindromes, and thinks them to be fair (which is a fancy word for nice). A *palindrome* is just an integer that reads the same backwards and forwards - so 6, 11 and 121 are all palindromes, while 10, 12, 223 and 2244 are not (even though 010=10, we don't consider leading zeroes when determining whether a number is a palindrome).

He recently became interested in squares as well, and formed the definition of a *fair and square* number - it is a number that is a palindrome **and** the *square of a palindrome* at the same time. For instance, 1, 9 and 121 are fair and square (being palindromes and squares, respectively, of 1, 3 and 11), while 16, 22 and 676 are **not** fair and square: 16 is not a palindrome, 22 is not a square, and while 676 is a palindrome and a square number, it is the square of 26, which is not a palindrome.

Now he wants to search for bigger fair and square numbers. Your task is, given an interval Little John is searching through, to tell him how many fair and square numbers are there in the interval, so he knows when he has found them all.

Solving this problem

Usually, Google Code Jam problems have 1 Small input and 1 Large input. This problem has 1 Small input and 2 Large inputs. Once you have solved the Small input, you will be able to download any of the two Large inputs. As usual, you will be able to retry the Small input (with a time penalty), while you will get only one chance at each of the Large inputs.

Input

The first line of the input gives the number of test cases, **T**. **T** lines follow. Each line contains two integers, **A** and **B** - the endpoints of the interval Little John is looking at.

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 number of fair and square numbers greater than or equal to **A** and smaller than or equal to **B**.

Limits

Small dataset

$1 \le \mathbf{T} \le 100$.
$1 \le \mathbf{A} \le \mathbf{B} \le 1000$.

First large dataset

$1 \le \mathbf{T} \le 10000$.
$1 \le \mathbf{A} \le \mathbf{B} \le 10^{14}$.

Second large dataset

$1 \le \mathbf{T} \le 1000$.
$1 \le \mathbf{A} \le \mathbf{B} \le 10^{100}$.

Sample

Input	Output
3 1 4 10 120 100 1000	Case #1: 2 Case #2: 0 Case #3: 2

Problem D. Treasure

Small input 20 points	Solve D-small
Large input 60 points	Solve D-large

Problem

Following an old map, you have stumbled upon the Dread Pirate Larry's secret treasure trove!

The treasure trove consists of **N** locked chests, each of which can only be opened by a key of a specific type. Furthermore, once a key is used to open a chest, it can never be used again. Inside every chest, you will of course find lots of treasure, and you might also find one or more keys that you can use to open other chests. A chest may contain multiple keys of the same type, and you may hold any number of keys.

You already have at least one key and your map says what other keys can be found inside the various chests. With all this information, can you figure out how to unlock all the chests?

For example, suppose the treasure trove consists of four chests as described below, and you began with exactly one key of type 1:

Chest Number	Key Type To Open Chest	Key Types Inside
1	1	None
2	1	1, 3
3	2	None
4	3	2

You can open all the chests in this example if you do them in the order 2, 1, 4, 3. If you start by opening chest #1 first, then you will have used up your only key, and you will be stuck.

Input

The first line of the input gives the number of test cases, **T**. **T** test cases follow. Each test case begins with a single line containing two positive integers **K** and **N**, representing the number of keys you start with and the number of chests you need to open.

This is followed by a line containing **K** integers, representing the types of the keys that you start with.

After that, there will be **N** lines, each representing a single chest. Each line will begin with integers $\mathbf{T_i}$ and $\mathbf{K_i}$, indicating the key type needed to open the chest and the number of keys inside the chest. These two integers will be followed by $\mathbf{K_i}$ more integers, indicating the types of the keys contained within the chest.

Output

For each test case, output one line containing "Case #x: C_1 C_2 ... $C_{\mathbf{N}}$", where x is the case number (starting from 1), and where C_i represents the index (starting from 1) of the i^{th} chest that you should open.

If there are multiple ways of opening all the chests, choose the "lexicographically smallest" way. In other words, you should choose to make C_1 as small as possible, and if there are multiple ways of making C_1 as small as possible, choose the one that makes C_2 as small as possible, and so on.

If there is no way to open all the chests, you should instead output one line containing "Case #x: IMPOSSIBLE".

Limits

$1 \le \mathbf{T} \le 25$.
$1 \le \mathbf{K}$.
All key types will be integers between 1 and 200 inclusive.

Small dataset

$1 \le \mathbf{N} \le 20$.
In each test case, there will be at most 40 keys altogether.

Large dataset

$1 \leq \mathbf{N} \leq 200$.
In each test case, there will be at most 400 keys altogether.

Sample

Input	Output
3 1 4 1 1 0 1 2 1 3 2 0 3 1 2 3 3 1 1 1 1 0 1 0 1 0 1 1 2 1 1 1	Case #1: 2 1 4 3 Case #2: 1 2 3 Case #3: IMPOSSIBLE

Problem A. Bullseye

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 11 points	Solve A-small
Large input 13 points	Solve A-large

Problem

Maria has been hired by the Ghastly Chemicals Junkies (GCJ) company to help them manufacture **bullseyes**. A **bullseye** consists of a number of concentric rings (rings that are centered at the same point), and it usually represents an archery target. GCJ is interested in manufacturing black-and-white bullseyes.

Maria starts with **t** millilitres of black paint, which she will use to draw rings of thickness 1cm (one centimetre). A ring of thickness 1cm is the space between two concentric circles whose radii differ by 1cm.

Maria draws the first black ring around a white circle of radius **r** cm. Then she repeats the following process for as long as she has enough paint to do so:

1. Maria imagines a white ring of thickness 1cm around the last black ring.
2. Then she draws a new black ring of thickness 1cm around that white ring.

Note that each "white ring" is simply the space between two black rings.

The area of a disk with radius 1cm is **π** cm^2. One millilitre of paint is required to cover area **π** cm^2. What is the maximum number of black rings that Maria can draw? Please note that:

- Maria only draws complete rings. If the remaining paint is not enough to draw a complete black ring, she stops painting immediately.
- There will always be enough paint to draw at least one black ring.

Input

The first line of the input gives the number of test cases, **T**. **T** test cases follow. Each test case consists of a line containing two space separated integers: **r** and **t**.

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 number of black rings that Maria can draw.

Limits

Small dataset

$1 \le \mathbf{T} \le 1000$.
$1 \le \mathbf{r}, \mathbf{t} \le 1000$.

Large dataset

$1 \le \mathbf{T} \le 6000$.
$1 \le \mathbf{r} \le 10^{18}$.
$1 \le \mathbf{t} \le 2 \times 10^{18}$.

Sample

Input

```
5
1 9
1 10
3 40
1 1000000000000000000
10000000000000000 1000000000000000000
```

Output

```
Case #1: 1
Case #2: 2
Case #3: 3
Case #4: 707106780
Case #5: 49
```

Problem B. Manage your Energy

Small input 12 points	Solve B-small
Large input 23 points	Solve B-large

Problem

You've got a very busy calendar today, full of important stuff to do. You worked hard to prepare and make sure all the activities don't overlap. Now it's morning, and you're worried that despite all of your enthusiasm, you won't have the energy to do all of this with full engagement.

You will have to manage your energy carefully. You start the day full of energy - **E** joules of energy, to be precise. You know you can't go below zero joules, or you will drop from exhaustion. You can spend any non-negative, integer number of joules on each activity (you can spend zero, if you feel lazy), and after each activity you will regain **R** joules of energy. No matter how lazy you are, however, you **cannot** have more than **E** joules of energy at any time; any extra energy you would regain past that point is wasted.

Now, some things (like solving Code Jam problems) are more important than others. For the **i**th activity, you have a value $\mathbf{v_i}$ that expresses how important this activity is to you. The *gain* you get from each activity is the value of the activity, multiplied by the amount of energy you spent on the activity (in joules). You want to manage your energy so that your total gain will be as large as possible.

Note that you *cannot* reorder the activities in your calendar. You just have to manage your energy as well as you can with the calendar you have.

Input

The first line of the input gives the number of test cases, **T**. **T** test cases follow. Each test case is described by two lines. The first contains three integers: **E**, the maximum (and initial) amount of energy, **R**, the amount you regain after each activity, and **N**, the number of activities planned for the day. The second line contains **N** integers $\mathbf{v_i}$, describing the values of the activities you have planned for today.

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 gain you can achieve by managing your energy that day.

Limits

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

Small dataset

$1 \leq \mathbf{E} \leq 5$.
$1 \leq \mathbf{R} \leq 5$.
$1 \leq \mathbf{N} \leq 10$.
$1 \leq \mathbf{v_i} \leq 10$.

Large dataset

$1 \leq \mathbf{E} \leq 10^7$.
$1 \leq \mathbf{R} \leq 10^7$.
$1 \leq \mathbf{N} \leq 10^4$.
$1 \leq \mathbf{v_i} \leq 10^7$.

Sample

Input	Output
3 5 2 2 2 1 5 2 2 1 2 3 3 4 4 1 3 5	Case #1: 12 Case #2: 12 Case #3: 39

In the first case, we can spend all 5 joules of our energy on the first activity (for a gain of 10), regain 2 and spend them on the second activity. In the second case, we spend 2 joules on the first activity, regain them, and spend 5 on the second. In the third case, our regain rate is equal to the maximum energy, meaning we always recover all energy after each activity - so we can spend full 3 joules on each activity.

Problem C. Good Luck

Small input 1 10 points	Solve C-small-1
Small input 2 31 points	Solve C-small-2

Problem

Maryam and Peiling have recently been practicing a new number trick, and they need your help to get it right. The trick goes as follows: Maryam starts by picking **N** independent random integer numbers, each between 2 and **M**, inclusive, appearing with equal probability, and writes them down on **N** cards, one number per card. Note that some numbers might be equal. Then, she repeats the following **K** times: take a random subset of cards (each card is taken with probability 0.5), and write down the product of the numbers on those cards. Having done all that, she shows all **K** products to Peiling, and Peiling's goal is to guess what the original **N** numbers were, knowing just **N**, **M**, and the products.

An example game with **N**=3, **M**=4, **K**=4 might go like this: first, Maryam picks 3 random numbers between 2 and 4, inclusive - let's say she randomly chose $\mathbf{A}_1$=3, $\mathbf{A}_2$=3 and $\mathbf{A}_3$=4. Then, she calculates four products of random subsets of those three numbers. For example, let's say those products are $\mathbf{A}_1$*$\mathbf{A}_2$=9, $\mathbf{A}_3$=4, $\mathbf{A}_1$*$\mathbf{A}_2$*$\mathbf{A}_3$=36, and 1=1 (the last product has no numbers in it, so it's equal to 1). Peiling receives numbers 9,4,36,1 from her, and she's also told that **N**=3 and **M**=4. In this case, just seeing the number 36 is enough to find what the original numbers were, since the only way to represent that as a product of up to 3 numbers, each up to 4, is 3*3*4. So Peiling says that the original numbers were 3, 3 and 4, and the audience is impressed.

In some other cases, guessing the original numbers is not as simple. For example, it might happen that all products are equal to 1. In that case there is no way to know anything about the hidden numbers, so Peiling cannot always be right. However, Peiling knows that Maryam follows the procedure exactly as described above: she selects the first **N** numbers as independent uniform integers between 2 and **M**, and then selects **K** independent random subsets, picking each number into each subset independently with probability 0.5. Help Peiling use that knowledge to make better guesses!

Solving this problem

This problem is a bit unusual for Code Jam. You will be given **R** independent sets of **K** numbers each, and should print an answer for each set — this part is as usual. However, you don't need to get all of your answers right! Your solution will be considered correct if answers for at least **X** sets are correct, with the value of **X** given in the Limits for the given input, below. However, you must follow the output format, even for sets in which your answer doesn't turn out to be correct. The *only* thing that can be wrong on any sets, yet still allow you to be judged correct, is the digits you output; but there should still be exactly **N** digits printed for each case, and each digit must be between **2** and **M**.

This problem involves randomness, and thus it might happen that even the best possible solution doesn't make **X** correct guesses (remember the situation when all products are equal to 1?) for a certain input. Because of that, this problem doesn't have a Large input, but instead has two Small inputs. That means you can try again if you think you got unlucky. You may only attempt to solve the second Small input once you have solved the first one. Otherwise, both Small inputs work in the same way as Small inputs for any other problem: you may try multiple times, and there is a 4-minute penalty for incorrect submissions if you later solve that input, even if the only reason you got it wrong was chance.

Good luck!

Input

The first line of the input gives the number of test cases, **T**, which is always equal to 1. The second line of the input file contains four space-separated integers **R**, **N**, **M** and **K**, in that order. The next **R** lines describe one set of **K** products each. Each of those lines contains **K** space-separated integers — the products that Maryam passes to Peiling. It is guaranteed that all sets in the input are generated independently randomly according to the procedure from the problem statement.

Output

On the first line, output "Case #1:". On each of the next **R** lines output **N** digits — your guess for Maryam's hidden numbers for the corresponding set of

products. You can print the numbers for each set in any order, but there must be exactly **N** digits, each between 2 and **M**, inclusive (note that M<10, so none of the numbers will be more than one digit). Do not put spaces between the digits.

Limits

First Small dataset

T = 1.
R = 100.
N = 3.
M = 5.
K = 7.
You need to get at least **X**=50 sets right.

Second Small dataset

T = 1.
R = 8000.
N = 12.
M = 8.
K = 12.
You need to get at least **X**=1120 sets right.

Sample

Input	Output
1 2 3 4 4 9 4 36 1 1 1 1 1	Case #1: 343 222

Note

The sample input doesn't follow the limitations for either input. In the sample input, you need to get at least **X**=1 sets right.

In the sample input, Maryam picked the numbers 3, 3, 4 the first time, and the numbers 2, 4, 4 the second time. In the sample output, Peiling guessed correctly the first time, but not the second time.

Round 1B 2013

Submissions

Osmos

10pt	Not attempted **4627/7207 users** correct (64%)
12pt	Not attempted **3501/4538 users** correct (77%)

Falling Diamonds

14pt	Not attempted **935/1857 users** correct (50%)
28pt	Not attempted **522/714 users** correct (73%)

Garbled Email

12pt	Not attempted **444/894 users** correct (50%)
24pt	Not attempted **255/345 users** correct (74%)

Top Scores

dolphinigle	100
K.A.D.R	100
blmarket	100
rng..58	100
Seyaua	100
bmerry	100
jcn	100
chokudai	100
IvanRomanov	100
neal.wu	100

Problem A. Osmos

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 10 points	Solve A-small
Large input 12 points	Solve A-large

Problem

Armin is playing Osmos, a physics-based puzzle game developed by Hemisphere Games. In this game, he plays a "mote", moving around and absorbing smaller motes.

A "mote" in English is a small particle. In this game, it's a thing that absorbs (or is absorbed by) other things! The game in this problem has a similar idea to Osmos, but does not assume you have played the game.

When Armin's mote absorbs a smaller mote, his mote becomes bigger by the smaller mote's size. Now that it's bigger, it might be able to absorb even more motes. For example: suppose Armin's mote has size 10, and there are other motes of sizes 9, 13 and 19. At the start, Armin's mote can only absorb the mote of size 9. When it absorbs that, it will have size 19. Then it can only absorb the mote of size 13. When it absorbs that, it'll have size 32. Now Armin's mote can absorb the last mote.

Note that Armin's mote can absorb another mote if and only if the other mote is *smaller*. If the other mote is the same size as his, his mote can't absorb it.

You are responsible for the program that creates motes for Armin to absorb. The program has already created some motes, of various sizes, and has created Armin's mote. Unfortunately, given his mote's size and the list of other motes, it's possible that there's no way for Armin's mote to absorb them all.

You want to fix that. There are two kinds of operations you can perform, in any order, any number of times: you can add a mote of any positive integer size to the game, or you can remove any one of the existing motes. What is the minimum number of times you can perform those operations in order to make it possible for Armin's mote to absorb every other mote?

For example, suppose Armin's mote is of size 10 and the other motes are of sizes [9, 20, 25, 100]. This game isn't currently solvable, but by adding a mote of size 3 and removing the mote of size 100, you can make it solvable in only 2 operations. The answer here is 2.

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 gives the size of Armin's mote, **A**, and the number of other motes, **N**. The second line contains the **N** sizes of the other motes. All the mote sizes given will be integers.

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 operations needed to make the game solvable.

Limits

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

Small dataset

$1 \leq \mathbf{A} \leq 100$.
$1 \leq$ all mote sizes ≤ 100.
$1 \leq \mathbf{N} \leq 10$.

Large dataset

$1 \leq \mathbf{A} \leq 10^6$.
$1 \leq$ all mote sizes $\leq 10^6$.
$1 \leq \mathbf{N} \leq 100$.

Sample

Input	Output
4 2 2	Case #1: 0 Case #2: 1

```
2 1           Case #3: 2
2 4           Case #4: 4
2 1 1 6
10 4
25 20 9 100
1 4
1 1 1 1
```

Notes

Although the size of motes is limited in the input files, Armin's mote may grow larger than the provided limits by absorbing other motes.

Osmos was created by Hemisphere Games. Hemisphere Games does not endorse and has no involvement with Google Code Jam.

Round 1B 2013

A. Osmos
B. Falling Diamonds
C. Garbled Email

Contest Analysis
Questions asked

Problem B. Falling Diamonds

Small input 14 points	Solve B-small
Large input 28 points	Solve B-large

Problem

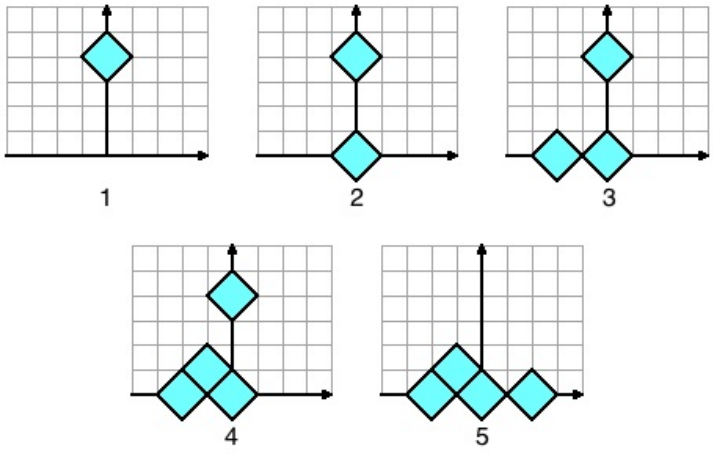
Diamonds are falling from the sky. People are now buying up locations where the diamonds can land, just to own a diamond if one does land there. You have been offered one such place, and want to know whether it is a good deal.

Diamonds are shaped like, you guessed it, diamonds: they are squares with vertices (X-1, Y), (X, Y+1), (X+1, Y) and (X, Y-1) for some X, Y which we call the center of the diamond. All the diamonds are always in the X-Y plane. X is the horizontal direction, Y is the vertical direction. The ground is at Y=0, and positive Y coordinates are above the ground.

The diamonds fall one at a time along the Y axis. This means that they start at (0, Y) with Y very large, and fall vertically down, until they hit either the ground or another diamond.

When a diamond hits the ground, it falls until it is buried into the ground up to its center, and then stops moving. This effectively means that all diamonds stop falling or sliding if their center reaches Y=0.

When a diamond hits another diamond, vertex to vertex, it can start sliding down, without turning, in one of the two possible directions: down and left, or down and right. If there is no diamond immediately blocking either of the sides, it slides left or right with equal probability. If there is a diamond blocking one of the sides, the falling diamond will slide to the other side until it is blocked by another diamond, or becomes buried in the ground. If there are diamonds blocking the paths to the left and to the right, the diamond just stops.



Consider the example in the picture. The first diamond hits the ground and stops when halfway buried, with its center at (0, 0). The second diamond may slide either to the left or to the right with equal probability. Here, it happened to go left. It stops buried in the ground next to the first diamond, at (-2, 0). The third diamond will also hit the first one. Then it will either randomly slide to the right and stop in the ground, or slide to the left, and stop between and above the two already-placed diamonds. It again happened to go left, so it stopped at (-1, 1). The fourth diamond has no choice: it will slide right, and stop in the ground at (2, 0).

Input

The first line of the input gives the number of test cases, **T**. **T** lines follow. Each line contains three integers: the number of falling diamonds **N**, and the position **X, Y** of the place you are interested in. Note the place that you are interested in buying does not have to be at or near the ground.

Output

For each test case output one line containing "Case #x: p", where x is the case number (starting from 1) and p is the probability that one of the **N** diamonds will fall so that its center ends up exactly at (**X**, **Y**). The answer will be considered correct if it is within an absolute error of 10^{-6} away from the correct answer. See the FAQ for an explanation of what that means, and what formats of floating-point numbers we accept.

Limits

1 ≤ **T** ≤ 100.
-10,000 ≤ **X** ≤ 10,000.
0 ≤ **Y** ≤ 10,000.
X + **Y** is even.

Small dataset

1 ≤ **N** ≤ 20.

Large dataset

1 ≤ **N** ≤ 10^6.

Sample

Input	Output
7 1 0 0 1 0 2 3 0 0 3 2 0 3 1 1 4 1 1 4 0 2	Case #1: 1.0 Case #2: 0.0 Case #3: 1.0 Case #4: 0.75 Case #5: 0.25 Case #6: 0.5 Case #7: 0.0

Problem C. Garbled Email

Small input 12 points	Solve C-small
Large input 24 points	Solve C-large

Problem

Gagan just got an email from her friend Jorge. The email contains important information, but unfortunately it was corrupted when it was sent: all of the spaces are missing, and after the removal of the spaces, some of the letters have been changed to other letters! All Gagan has now is a string **S** of lower-case characters.

You know that the email was originally made out of words from the dictionary described below. You also know the letters were changed after the spaces were removed, and that the difference between the indices of any two letter changes is not less than 5. So for example, the string "code jam" could have become "codejam", "dodejbm", "zodejan" or "cidejab", but not "kodezam" (because the distance between the indices of the "k" change and the "z" change is only 4).

What is the minimum number of letters that could have been changed?

Dictionary

In order to solve this problem, you'll need an extra file: a special dictionary that you can find at https://code.google.com/codejam/contest/static/garbled_email_dictionary.txt. It is not a dictionary from any natural language, though it does contain some English words. Each line of the dictionary contains one word. The dictionary file should be 3844492 bytes in size, contain 521196 words, start with the word "a", and end with the word "zymuznh".

When you're submitting the code you used to solve this problem, you shouldn't include the dictionary. As usual, however, you must submit all code you used to solve the problem.

Note that if you are using Windows and want to look at the dictionary file, you should avoid Notepad, and instead use WordPad or another piece of software, or else all the words might appear on the same line.

Input

The first line of the input gives the number of test cases, **T**. **T** test cases follow. Each test case consists of a single line containing a string **S**, consisting of lower-case characters a-z.

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 letters that could have been changed in order to make **S**.

Limits

S is valid: it is possible to make it using the method described above.

Small dataset

$1 \le \mathbf{T} \le 20$.
1 ≤ length of **S** ≤ 50.

Large dataset

$1 \le \mathbf{T} \le 4$.
1 ≤ length of **S** ≤ 4000.

Sample

Input	Output
4 codejam cxdejax cooperationaabea jobsinproduction	Case #1: 0 Case #2: 2 Case #3: 1 Case #4: 1

Explanation

"code" and "jam" both appear in the dictionary. Although "cooperation" is an English word, it doesn't appear in the dictionary; "aabea" does.

Round 1C 2013

A. Consonants

B. Pogo

C. The Great Wall

Contest Analysis

Questions asked 1

Submissions

Consonants

8pt | Not attempted **4290/4819 users** correct (89%)

20pt | Not attempted **1538/3763 users** correct (41%)

Pogo

10pt | Not attempted **2522/3113 users** correct (81%)

25pt | Not attempted **121/637 users** correct (19%)

The Great Wall

9pt | Not attempted **930/1253 users** correct (74%)

28pt | Not attempted **74/330 users** correct (22%)

Top Scores

staniek	100
eatmore	100
turbin	100
ir5	100
tkociumaka	100
Gerald.	100
DCLXVI	100
random.johnnyh	100
jamu	100
nicesap	100

Problem A. Consonants

Small input 8 points	Solve A-small
Large input 20 points	Solve A-large

Problem

In English, there are 26 letters that are either **vowels** or **consonants**. In this problem, we consider **a, e, i, o,** and **u** to be vowels, and the other 21 letters to be consonants.

A tribe living in the Greatest Colorful Jungle has a tradition of naming their members using English letters. But it is not easy to come up with a good name for a new member because it reflects the member's social status within the tribe. It is believed that the less common the name he or she is given, the more socially privileged he or she is.

The leader of the tribe is a professional linguist. He notices that hard-to-pronounce names are uncommon, and the reason is that they have too many **consecutive consonants**. Therefore, he announces that the social status of a member in the tribe is determined by its **n**-value, which is the number of substrings with at least **n** consecutive consonants in the name. For example, when **n** = 3, the name "quartz" has the **n**-value of 4 because the substrings **quartz**, **uartz**, **artz**, and **rtz** have at least 3 consecutive consonants each. A greater **n**-value means a greater social status in the tribe. Two substrings are considered different if they begin or end at a different point (even if they consist of the same letters), for instance "tsetse" contains 11 substrings with two consecutive consonants, even though some of them (like "**tse**tse" and "tse**tse**") contain the same letters.

All members in the tribe must have their names and **n** given by the leader. Although the leader is a linguist and able to ensure that the given names are meaningful, he is not good at calculating the **n**-values. Please help the leader determine the **n**-value of each name. Note that different names may have different values of **n** associated with them.

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 gives the name of a member as a string of length **L**, and an integer **n**. Each name consists of one or more lower-case English letters.

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 **n**-value of the member's name.

Limits

$1 \leq \mathbf{T} \leq 100$.
$0 < \mathbf{n} \leq \mathbf{L}$.

Small dataset

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

Large dataset

$1 \leq \mathbf{L} \leq 10^6$.
The input file will be no larger than 6MB.

Sample

Input	Output
4 quartz 3 straight 3 gcj 2 tsetse 2	Case #1: 4 Case #2: 11 Case #3: 3 Case #4: 11

Round 1C 2013

A. Consonants

B. Pogo

C. The Great Wall

Contest Analysis

Questions asked 1

Problem B. Pogo

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

Problem

You have just got the best gift ever, a Pogo stick. The pogo stick is something you use to jump off the ground while standing on it.

This Pogo stick is a special one: the first jump will move you a distance of 1 unit, the second jump will move you 2 units, the third jump will move you 3 units and so on. You can jump in only four directions using this stick: north (increasing y), south (decreasing y), east (increasing x) or west (decreasing x).

Now you want to play a game in your backyard, which we model as an infinite plane. You are standing with your stick in at point (0, 0) and you want to go to point (**X**, **Y**).

The point (**X**, **Y**) will never be (0, 0), and it will always be reachable from your starting point.

Check the output section carefully, because the required outputs for the small and large datasets are not the same.

Input

The first line of the input gives the number of test cases, **T**. **T** test cases follow, one per line. Each line consists of 2 integers separated by a single space, **X** and **Y**, the coordinates of the point you want to reach.

Output

For each test case, output one line containing "Case #x: y", where x is the case number (starting from 1) and y is a string represents the directions of the moves, for example if you are going to move north then south then east then west, this string should be NSEW.

For the small dataset, the output is considered correct if it does not take more than 500 moves to reach the destination in each test case.

For the large dataset, the output is considered correct if it reaches the destination point in the minimum possible number of moves.

If there are multiple correct solutions, print any of them.

Limits

Small dataset

$1 \le \mathbf{T} \le 50$.
$0 \le |\mathbf{X}|, |\mathbf{Y}| \le 100$.

Large dataset

$1 \le \mathbf{T} \le 100$.
$0 \le |\mathbf{X}|, |\mathbf{Y}| \le 10^6$.

Sample

Input	Output
2 3 4 -3 4	Case #1: ENWSEN Case #2: ENSWN

The output for the first sample test case will not be considered correct if it is in the large dataset, because the number of moves is not the minimum. WNSEN would be a correct output for this test case if it were in the large dataset.

Powered by

Google Cloud Platform

Problem C. The Great Wall

Problem

You are studying the history of the Great Wall of China, which was built by the Chinese to protect against military incursions from the North. For the purposes of this problem, the Great Wall stretches from infinity in the East to minus infinity in the West. As this is a lot of distance to cover, the Great Wall was not built at once. Instead, for this problem we assume that the builder used a reactive strategy: whenever a part of the border was attacked successfully, the Wall on this part of the border would be raised to the height sufficient to stop an identical attack in the future.

The north border of China was frequently attacked by nomadic tribes. For the purposes of this problem, we assume that each tribe attacks the border on some interval with some strength *S*. In order to repel the attack, the Wall must have height *S* all along the defended interval. If even a short stretch of the Wall is lower than needed, the attack will breach the Wall at this point and succeed. Note that even a successful attack does not damage the Wall. After the attack, every attacked fragment of the Wall that was lower than *S* is raised to height *S* — in other words, the Wall is increased in the minimal way that would have stopped the attack. Note that if two or more attacks happened on the exact same day, the Wall was raised only after they all resolved, and is raised in the minimum way that would stop all of them.

Since nomadic tribes are nomadic, they did not necessarily restrict themselves to a single attack. Instead, they tended to move (either to the East or to the West), and periodically attack the Wall. To simplify the problem, we assume they moved with constant speed and attacked the Wall at constant intervals; moreover we assume that the strength with which a given tribe attacked the Wall changed by a constant amount after each attack (either decreased from attrition, or grew from experience).

Assuming that initially (in 250 BC) the Wall was nonexistent (i.e., of height zero everywhere), and given the full description of all the nomadic tribes that attacked the Wall, determine how many of the attacks were successful.

Input

The first line of the input gives the number of test cases, **T**. **T** test cases follow. Each test case begins with a line containing a single integer **N**: the number of the tribes attacking the Wall. **N** lines follow, each describing one tribe. The **i**th line contains eight integers $\mathbf{d_i}$, $\mathbf{n_i}$, $\mathbf{w_i}$, $\mathbf{e_i}$, $\mathbf{s_i}$, $\mathbf{delta\_d_i}$, $\mathbf{delta\_p_i}$ and $\mathbf{delta\_s_i}$ separated by spaces, describing a single nomadic tribe:

- $\mathbf{d_i}$ – the day of the tribe's first attack (where 1st January, 250BC, is considered day 0)
- $\mathbf{n_i}$ – the number of attacks from this tribe
- $\mathbf{w_i}$, $\mathbf{e_i}$ – the westmost and eastmost points respectively of the Wall attacked on the first attack
- $\mathbf{s_i}$ – the strength of the first attack
- $\mathbf{delta\_d_i}$ – the number of days between subsequent attacks by this tribe
- $\mathbf{delta\_p_i}$ – the distance this tribe travels to the east between subsequent attacks (if this is negative, the tribe travels to the west)
- $\mathbf{delta\_s_i}$ – the change in strength between subsequent attacks

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 number of attacks that succeed.

Limits

$1 \le \mathbf{T} \le 20$.
$0 \le \mathbf{d_i}$.
$1 \le \mathbf{delta\_d_i} \le 676060$.
$\mathbf{d_i} + (\mathbf{n_i} - 1) * \mathbf{delta\_d_i} \le 676060$.
$1 \le \mathbf{s_i} \le 10^6$.
$-10^5 \le \mathbf{delta\_s_i} \le 10^5$.
$\mathbf{s_i} + (\mathbf{n_i} - 1) * \mathbf{delta\_s_i} \ge 1$.

Small dataset

$1 \le \mathbf{N} \le 10$.
$1 \le \mathbf{n_i} \le 10$.
$-100 \le \mathbf{w_i} < \mathbf{e_i} \le 100$.
$-10 \le \mathbf{delta\_p_i} \le 10$.

Large dataset

$1 \le \mathbf{N} \le 1000$.
$1 \le \mathbf{n_i} \le 1000$.
$-10^6 \le \mathbf{w_i} < \mathbf{e_i} \le 10^6$.
$-10^5 \le \mathbf{delta\_p_i} \le 10^5$.

Sample

Input	Output
2 2 0 3 0 2 10 2 3 -2 10 3 2 3 8 7 2 0 3 1 2 0 5 10 2 8 0 0 3 0 1 7 1 2 2 3 3 0 5 1 1 4 0	Case #1: 5 Case #2: 6

In the first case, the first tribe attacks three times: on day 0 it hits the interval [0,2] at height 10, on day 2 it hits [3,5] at height 8 and on day 4 it hits [6,8] at height 6; all three attacks succeed. Then the second tribe attacks three times, each time at height 8 - on day 10 it hits [2,3] (this succeeds, for example at position 2.5, where the Wall has still height 0), on day 17 it hits [4,5] (this fails, the Wall is already of height 8 in the interval [3, 5], which covers [4, 5]), and on day 24 it hits [6,7] (this succeeds, as the Wall there was of height 6).

In the second case there are three tribes, and their attacks intermingle. The sequence is as follows:

- On day 0, Tribe 2 attacks [0,1] at height 7 and succeeds.
- On day 1, Tribe 1 attacks [0,5] at height 10, and Tribe 2 attacks [2,3] at height 9. Both attacks succeed (as they were simultaneous, the Wall built after the attack of the first tribe isn't there in time to stop the second tribe).
- On day 2, Tribe 2 attacks [4,5] at height 11 and succeeds (the Wall there was at height 10).
- On day 3, Tribe 1 attacks [8,13] at height 10 and succeeds. Simultaneously, Tribe 3 attacks [0,5] at height 1 and fails (there's a Wall of heights 10 and 11 there).
- On day 4 Tribe 3 attacks [4,9] at height 1 and succeeds (there was no Wall between 5 and 8).
- Finally, on day 5 Tribe 3 attacks [8,13] at height 1 and fails (since a Wall of height 10 is there).

Problem A. Ticket Swapping

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

Problem

The city has built its first subway line, with a grand total of **N** stations, and introduced a new way of paying for travel. Instead of just paying for one ticket and making an arbitrary journey, the price you pay is now based on *entry cards*.

When entering the subway, each passenger collects an *entry card*, which specifies the station the passenger entered at. When leaving the subway, the passenger has to give up the entry card, and is charged depending on the distance (in stations traveled) between the entry station specified on the entry card, and the exit station on which the entry card is surrendered. The payment depends on the distance between these stations as follows:

- if they are the same station, you don't pay;
- if they are adjacent stations, you pay **N** pounds;
- if the distance is two stations, you pay **2N - 1**: a charge **N** for the first stop and **N - 1** for the second;
- the third station costs **N-2** (so you pay **3N - 3** for a three-station-long trip), the fourth stop **N-3**, and the *i*th stop **N + 1-*i***;
- thus, if you travel from one end of the subway to the other (a distance of **N**-1 stations), you pay 2 pounds for the last station traveled, and a grand total of ($\mathbf{N}^2$ + **N** - 2) / 2 in total.

After introducing this system the city noticed their gains are not as big as they expected. They figured out this might be due to people swapping their entry cards — so, for instance, if one person enters at station **A**, travels two stations to **B** and exits, while another person enters at **B**, travels three stations to **C** and exits, they would normally pay (in total) **2N - 1 + 3N - 3 = 5N - 4**. But if the two people swapped their entry cards at station **B**, then the first one would travel for free (as he would surrender an entry card specifying the station **B** while exiting a station **B**, and so register a distance of zero); while the second person will exit at station **C** and surrender an entry card specifying station **A**, which is 5 stations away, and pays **5N - 10**, at a net loss of six pounds to the city!

The city now wants to learn how much they can possibly lose if this practice becomes widespread. We will consider only one direction (from station 1 to station **N**, passing through all the stations in order) of the subway, and only one train on this line. We assume a passenger travelling from **o** to **e** obtains an entry card at **o**, can swap her entry card any number of times with any other passengers anywhere between **o** and **e**, including swapping with people who leave at **o** or those who enter at **e**, and then exit the train at **e** with some entry card (it is necessary to surrender some entry card to exit the subway). We also assume the passenger will not exit the train in the meantime (that is, will not surrender the currently held card and obtain a new one).

You are given a map of traffic (specifying how many passengers travel this train from which station to which), and you should calculate the city's financial loss, assuming passengers swap their cards to maximize this loss.

Input

The first line of the input gives the number of test cases, **T**. **T** test cases follow. Each test case contains the number **N** of stops (the stops are numbered 1 to **N**), and the number **M** of origin-endpoint pairs given. The next **M** lines contain three numbers each: the origin stop $\mathbf{o_i}$, the end stop $\mathbf{e_i}$ and $\mathbf{p_i}$: the number of passengers that make this journey.

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 total loss the city can observe due to ticket swapping, modulo 1000002013.

Limits

$1 \le \mathbf{T} \le 20$.
$1 \le \mathbf{o_i} < \mathbf{e_i} \le \mathbf{N}$

Small dataset

$2 \le \mathbf{N} \le 100$.
$1 \le \mathbf{M} \le 100$.
$1 \le \mathbf{p_i} \le 100$.

Large dataset

$2 \le \mathbf{N} \le 10^9$.
$1 \le \mathbf{M} \le 1000$.
$1 \le \mathbf{p_i} \le 10^9$.

Sample

Input	Output
3 6 2 1 3 1 3 6 1 6 2 1 3 2 4 6 1 10 2 1 7 2 6 9 1	Case #1: 6 Case #2: 0 Case #3: 10

The first test case is the case described in the problem statement - two passengers meet at station 3 and swap tickets. In the second test case the two passengers don't meet at all, so they can't swap tickets (and so the city incurs no loss). In the third case, only one of the early passengers can swap tickets with the later passenger.

Problem B. Many Prizes

Small input 7 points	Solve B-small
Large input 13 points	Solve B-large

Problem

We're going to run a tournament with 2^{N} teams, and give out **P** identical prizes to the teams with ranks `0..P-1`.

The teams are numbered 0 through $2^{N}-1$. When team i and team j play against each other in a game, team i will win iff `i<j`.

The teams for a tournament are organized in some order, called the tournament's *tournament list*, which contains all 2^{N} teams in the tournament. The tournament list will affect which teams play each other, and in what order.

Your job will be to find the largest-numbered team that is *guaranteed* to win a prize, independent of how the tournament list is ordered; and to find the largest-numbered team that *could* win a prize, depending on how the tournament list is ordered.

Tournament Resolution

The tournament is conducted in **N** *rounds*.

Each team has a *record*: the list of the results of the games it has played so far. For example, if a team has played three games, and won the first, lost the second and won the third, its record is `[W, L, W]`. If a team has played zero games, its record is `[]`.

In each round, every team plays a game against a team with the same record. The first team in the tournament list with a particular record will play against the second team with that record; the third team with the same record will play against the fourth; and so on.

After **N** rounds, each team has a different record. The teams are ranked in reverse lexicographical order of their records; so `[W, W, W] > [W, W, L] > [W, L, W] ... > [L, L, L]`.

Here is an example of a tournament with **N**=3, and the tournament list `[2, 4, 5, 3, 6, 7, 1, 0]`, where the columns represent different rounds, and the teams are grouped by their records. The winner of each game in the example has been marked with a *.

```
Round 1    Round 2    Round 3    Final Result
                                 (best rank at top)
[]         [W]        [W,W]
2  *       2  *       2          0  [W,W,W]
4          3          0  *       2  [W,W,L]
                      [W,L]
5          6          3  *       3  [W,L,W]
3  *       0  *       6          6  [W,L,L]
           [L]        [L,W]
6  *       4  *       4          1  [L,W,W]
7          5          1  *       4  [L,W,L]
                      [L,L]
1          7          5  *       5  [L,L,W]
0  *       1  *       7          7  [L,L,L]
```

If we give out 4 prizes (**N**=3, **P**=4), the prizes will go to teams 0, 2, 3 and 6.

The largest-numbered team that was guaranteed to win a prize with **N**=3, **P**=4, independent of the order of the tournament list, was team 0: this tournament list demonstrated that it's possible for team 1 *not* to win a prize, and it turns out that team 0 will always win one, regardless of the order of the tournament list.

The largest-numbered team that could win a prize with **N**=3, **P**=4, depending on how the tournament list was ordered, was team 6: this tournament list demonstrated that it's possible for team 6 to win a prize, and it turns out that team 7 will never win one, regardless of the order of the tournament list.

Input

The first line of the input gives the number of test cases, **T**. **T** test cases follow. Each test case consists of two space-separated integers: **N**, which indicates the tournament has 2^{N} teams, and **P**, the number of prizes.

Output

For each test case, output one line containing "Case #x: y z", where x is the case number (starting from 1), y is the largest-numbered team that is *guaranteed* to win a prize, independent of how the tournament list is ordered; and z is the largest-numbered team that *could* win a prize, depending on how the tournament list is ordered.

Limits

$1 \leq \mathbf{T} \leq 100$.
$1 \leq \mathbf{P} \leq 2^{\mathbf{N}}$.

Small dataset

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

Large dataset

$1 \leq \mathbf{N} \leq 50$.

Sample

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

Problem C. Erdős-Szekeres

Small input 9 points	Solve C-small
Large input 15 points	Solve C-large

Problem

Given a list **X**, consisting of the numbers (1, 2, ..., **N**), an *increasing subsequence* is a subset of these numbers which appears in increasing order, and a *decreasing subsequence* is a subset of those numbers which appears in decreasing order. For example, (5, 7, 8) is an increasing subsequence of (4, 5, 3, 7, 6, 2, 8, 1).

Nearly 80 years ago, two mathematicians, Paul Erdős and George Szekeres proved a famous result: **X** is guaranteed to have either an increasing subsequence of length at least sqrt(**N**) or a decreasing subsequence of length of at least sqrt(**N**). For example, (4, 5, 3, 7, 6, 2, 8, 1) has a decreasing subsequence of length 4: (5, 3, 2, 1).

I am teaching a combinatorics class, and I want to "prove" this theorem to my class by example. For every number X[i] in the sequence, I will calculate two values:

- A[i]: The length of the longest increasing subsequence of **X** that includes X[i] as its largest number.
- B[i]: The length of the longest decreasing subsequence of **X** that includes X[i] as its largest number.

The key part of my proof will be that the pair (A[i], B[i]) is different for every i, and this implies that either A[i] or B[i] must be at least sqrt(**N**) for some i. For the sequence listed above, here are all the values of A[i] and B[i]:

```
 i  |  X[i]  |  A[i]  |  B[i]
----+--------+--------+--------
 0  |   4    |   1    |   4
 1  |   5    |   2    |   4
 2  |   3    |   1    |   3
 3  |   7    |   3    |   4
 4  |   6    |   3    |   3
 5  |   2    |   1    |   2
 6  |   8    |   4    |   2
 7  |   1    |   1    |   1
```

I came up with a really interesting sequence to demonstrate this fact with, and I calculated A[i] and B[i] for every i, but then I forgot what my original sequence was. Given A[i] and B[i], can you help me reconstruct **X**?

X should consist of the numbers (1, 2, ..., **N**) in some order, and if there are multiple sequences possible, you should choose the one that is lexicographically smallest. This means that X[0] should be as small as possible, and if there are still multiple solutions, then X[1] should be as small as possible, and so on.

Input

The first line of the input gives the number of test cases, **T**. **T** test cases follow, each consisting of three lines.

The first line of each test case contains a single integer **N**. The second line contains **N** positive integers separated by spaces, representing A[0], A[1], ..., A[**N**-1]. The third line also contains **N** positive integers separated by spaces, representing B[0], B[1], ..., B[**N**-1].

Output

For each test case, output one line containing "Case #x: ", followed by X[0], X[1], ... X[**N**-1] in order, and separated by spaces.

Limits

$1 \le \mathbf{T} \le 30$.
It is guaranteed that there is at least one possible solution for **X**.

Small dataset

$1 \le \mathbf{N} \le 20$.

Large dataset

1 ≤ **N** ≤ 2000.

Sample

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

Problem D. Multiplayer Pong

Small input 12 points	Solve D-small
Large input 25 points	Solve D-large

Problem

Two teams of players play pong. Pong is a simple computer game, where each player controls a paddle (which we assume to be a point), and a little ball bounces back and forth. The players in one team are required to bounce the ball in a fixed cyclic order (so in a three-player team, the first one to touch the ball would be P1, then P2, then P3 and only then P1 again), until one of the players doesn't manage to bounce it, at which point the ball leaves the playing field and this player's team loses.

To be more precise: the playing field is a rectangle of size **A**x**B**. On each vertical wall (of length **A**) there are a number of paddles, one for each player of the team guarding this wall. Each paddle is a point. All the paddles of the players on one team move vertically at the same speed (in units per second), and can pass each other freely. There is also a ball, for which we are given its initial position (horizontal and vertical, counted from the lower-left corner) and initial speed (horizontal and vertical, again in units per second). The players are allowed to choose the initial positioning of their paddles on their vertical walls knowing the initial position of the ball. Whenever the ball reaches a horizontal wall, it bounces off (with the angle of incidence equal to the angle of reflection). Whenever it reaches a vertical end of the field, if the paddle of the player who is supposed to touch the ball now is there, it bounces off, while if there isn't, the team of the player whose paddle was supposed to be there loses.

The game can take quite a long time, with the players bouncing the ball back and forth. Your goal is to determine the final result (assuming all players play optimally).

Input

The first line of the input gives the number of test cases, **T**. **T** test cases follow. Each test case consists of four lines. The first line contains two integers, **A** and **B**, describing the height and width of the playing field. The second line contains two integers, **N** and **M**, describing the sizes of the two teams: **N** is the number of players on the team with paddles on the **X** = 0 wall, and **M** is the number of players on the team with paddles on **X** = **B** wall. The third line contains two integers, **V** and **W**, describing the speed of the paddles of players in the first and second team, respectively. The fourth line contains four integers: **Y**, **X**, $\mathbf{V_Y}$ and $\mathbf{V_X}$, describing the initial position (vertical and horizontal) and initial speed of the ball (the ball moves by $\mathbf{V_Y}$ units up and $\mathbf{V_X}$ to the right each second, until it bounces).

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 the three possible outputs: "DRAW" (if the game can proceed forever), "LEFT z", if the team with paddles on **x** = 0 wins, and the opposing team can bounce the ball at most z times, or "RIGHT z" if the team with paddles on **X** = **B** wins and the opposing team can bounce the ball at most z times.

Limits

$1 \le \mathbf{T} \le 100$.
$0 < \mathbf{X} < \mathbf{B}$
$0 < \mathbf{Y} < \mathbf{A}$

Small dataset

$1 \le \mathbf{N}, \mathbf{M} \le 10^6$
$1 \le \mathbf{V}, \mathbf{W} \le 10^{12}$
$-10^{12} \le \mathbf{V_Y} \le 10^{12}$
$-10^6 \le \mathbf{V_X} \le 10^6$
$2 \le \mathbf{A}, \mathbf{B} \le 10^6$

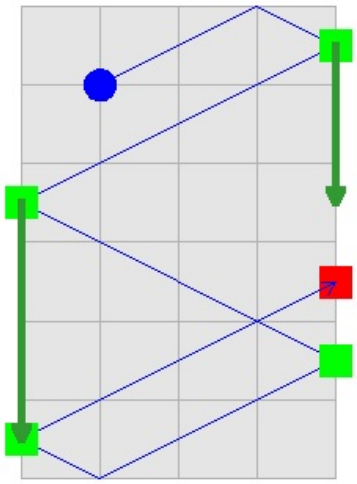
Large dataset

$1 \le \mathbf{N}, \mathbf{M} \le 10^{100}$
$1 \le \mathbf{V}, \mathbf{W} \le 10^{100}$

$-10^{100} \le \mathbf{V_Y}, \mathbf{V_X} \le 10^{100}$
$2 \le \mathbf{A}, \mathbf{B} \le 10^{100}$

Sample

Input	Output
4 6 4 1 2 3 1 5 1 4 8 12 3 3 1 2 3 1 1 2 4 12 3 1 3 3 1 1 1 2 4 12 2 1 2 10 2 3 1 13 4	Case #1: LEFT 2 Case #2: DRAW Case #3: LEFT 3 Case #4: RIGHT 11



The picture depicts the gameplay in the first sample case. The ball bounces off the right wall at time 0.375 (the first RIGHT player intercepts it, for instance by beginning with her paddle there and not moving it), then off the left wall at 0.875 (the LEFT player bounces it), again on the right at time 1.375 (the second RIGHT player can position his paddle at the bounce point), again on the left (where the LEFT player gets just in time to catch it — she covers the three units of distance exactly in one second in which she needs to get there) and then hits the right wall too far for the first RIGHT player to get there. Note the second RIGHT player could catch the ball, but is not allowed by the rules to do so. Also note that if RIGHT team had one player more, she could bounce the ball, and then LEFT would lose — the ball would come too far up for the single LEFT player to get there in time.

Round 3 2013

A. Cheaters

B. Rural Planning

C. Are We Lost Yet?

D. Observation Wheel

Contest Analysis

Questions asked

Problem A. Cheaters

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 7 points	Solve A-small
Large input 10 points	Solve A-large

Problem

You've been playing roulette for a while in a local casino. Roulette is a simple casino game in which multiple players place bets on one or more numbers between 0 and 36 (inclusive). Next, a wheel is spun in one direction with a ball spinning in the other direction. The roulette wheel contains the same numbers 0 to 36. Some real roulette wheels also have a space labeled 00, but ours does not. Eventually, the ball falls on one of the numbers. If a player placed a bet on that particular number, he receives 36 times his bet (so the profit of that bet is 35 times the bet). All bets placed on other numbers lose.

Unfortunately, luck hasn't been on your side, and you have been losing all night long. At one point, you started to wonder whether the roulette game was fair or not, and after observing the game some more, you noticed a pattern that must be profitable for the casino: the ball always lands on one of the numbers that has the least total money bet on it! If multiple numbers tie for the least total money bet, the ball lands on one of those uniformly at random.

Of course, you'll be notifying the authorities about this foul play, but first you want to win your money back by exploiting your new-found knowledge. To do so, you wait until all other players have placed their bets and then place bets of your own. Unfortunately, you have a limited budget left, so you cannot bet more than that. You are allowed to bet on zero or more different numbers, and each of those bets can be any positive integer amount (perhaps with different amounts for different numbers), so as long as the sum of your bets does not exceed your budget. What is the maximum expected profit you can make?

Input

The first line of input gives the number of cases, **T**. **T** test cases follow. Each test case consists of two lines. The first line contains two integers: the budget you still have, **B**, and the number of numbers other players have placed bets on, **N**. The second line contains **N** integers $\mathbf{X_i}$, the total amounts of money bet by other players on each of those different numbers.

Output

For each test case, output one line containing "Case #x: " followed by the maximum expected profit that you make if you place your bets optimally. A profit 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 floating-point numbers we accept.

Limits

1 <= **T** <= 100.
1 <= **N** <= 37.

Small dataset

1 <= **B, $\mathbf{X_i}$** <= 1,000.

Large dataset

1 <= **B, $\mathbf{X_i}$** <= 10^{12}.

Sample

Input	Output
3 100 1 10 34 3 5 6 7 34 4 1 1 10 10	Case #1: 0 Case #2: 2 Case #3: 0.9428571429

In example 2, bet 1 on each of the 34 empty numbers for a guaranteed payback of 36, and a profit of 36 - 34 = 2. In example 3, bet 1 on each of the 33 empty numbers, so that you win 36 with probability 33/35. The gives an expected profit of 33/35 * 36 - 33.

Problem B. Rural Planning

Small input 9 points	Solve B-small
Large input 13 points	Solve B-large

Problem

You have recently purchased a nice big farmyard, and you would like to build a fence around it. There are already **N** fence posts in your farmyard.

You will add lengths of fence in straight lines connecting the fence posts. Unfortunately, for reasons you don't fully understand, your lawyers insist you actually have to use *all* the fence posts, or things will go bad.

In this problem, the posts will be represented as points in a 2-dimensional plane. You want to build the fence by ordering the posts in some order, and then connecting the first with the second, second with third, and finally the last one with the first. The fence segments you create should be a polygon without self-intersections. That is, at each fence-post there are only two fence segments, and at every other point there is at most one fence segment.

Now that's easy, but you also actually want to preserve the fact your farmyard is big! It's not really fun to wall off most of your farmyard with the fences. So you would like to create the fence in such a way that the enclosed area is *more* than half of the maximum area you could enclose if you were allowed not to use all the posts.

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 the number **N** of posts. The posts are numbered from 0 to **N** - 1. Each of the next **N** lines contains two integers $\mathbf{X_i}$ and $\mathbf{Y_i}$ separated by a single space: the coordinates of the **i**-th post.

Output

For each test case, output one line containing "Case #x: ", where x is the case number (starting from 1), followed by **N** distinct integers from 0 to **N** - 1, separated by spaces. They are the numbers of the posts, in either clockwise or counter-clockwise direction, that you will use to build the fence. Note that the first and last posts are connected.

If there are multiple solutions, print any of them.

Limits

The posts will be at **N** unique points, and will not all lie on the same line.

Small dataset

$1 \le \mathbf{T} \le 100$
$3 \le \mathbf{N} \le 10$
$-100 \le \mathbf{X_i}, \mathbf{Y_i} \le 100$

Large dataset

$1 \le \mathbf{T} \le 30$
$3 \le \mathbf{N} \le 1000$
$-50000 \le \mathbf{X_i}, \mathbf{Y_i} \le 50000$

Sample

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

```
3
0 0
1 0
0 1
```

In the first test case, there are three polygons we can construct, and two of them have a large enough area — the ones described by sequences 0 1 2 3 and 0 2 1 3. The polygon described by 0 1 3 2 would be too small. In the second test case, we have make sure the polygon does not intersect itself, so, for instance, 0 1 2 3 4 or 0 1 3 4 2 would be bad. In the third case, any order describes the same triangle and is fine.

Round 3 2013

A. Cheaters
B. Rural Planning
C. Are We Lost Yet?
D. Observation Wheel

Contest Analysis
Questions asked

Problem C. Are We Lost Yet?

Small input
12 points
Solve C-small

Large input
18 points
Solve C-large

Problem

It is time for the Google Code Jam Finals, and we all want to be there! Unfortunately, a few of us accidentally ended up going to Mountain View instead of the correct location: London, England. But don't worry - we can take the free Google shuttle service from Mountain View to London!

The shuttle service consists of **M** one-way routes connecting pairs of cities. For every route, you know from which city and to which city it's going, but unfortunately you do not know exactly how long these routes are. Instead, for every route, you only know that its length can be any integer value from $\mathbf{a_i}$ to $\mathbf{b_i}$, inclusive.

I have taken Google shuttles many times before, so I have suggested a path of routes from Mountain View to London. But you worry that my path-finding skills are not as good as yours, and you want to check my work.

Given the path I am suggesting, could it possibly be a shortest path from Mountain View to London? If not, what is the ID of the first shuttle route on my path that is definitely not part of a shortest path (assuming that all previous shuttle routes have been taken according to the path I suggested)?

For example, suppose we have the following list of shuttle routes:

ID	Start City	Destination City	Shuttle Length
1	Mountain View	London	[100, 1000]
2	Mountain View	Paris	[500, 5000]
3	Paris	London	[400, 600]
4	Paris	Moscow	[500, 5000]
5	Moscow	London	[1, 10000]

I suggest the path Mountain View -> Paris -> Moscow -> London. The true shortest path might either be the direct route from Mountain View to London, or the path Mountain View -> Paris -> London. This means that the second route on my path (Paris -> Moscow) was the first one that is definitely not part of a shortest path.

Input

The first line of the input gives the number of test cases, **T**. **T** test cases follow. Each test begins with a line containing three positive integers **N**, **M**, and **P**. **N** represents the total number of cities (cities are numbered from 1 to **N**), **M** represents the total number of shuttle routes, and **P** represents the number of shuttle routes on my path from Mountain View (city #1) to London (city #2).

This is followed by **M** lines, each consisting of four integers, $\mathbf{u_i}$, $\mathbf{v_i}$, $\mathbf{a_i}$, $\mathbf{b_i}$. Each line represents the fact that there is a one-way shuttle route from city $\mathbf{u_i}$ to city $\mathbf{v_i}$, and you know that its length can be any integer value from $\mathbf{a_i}$ to $\mathbf{b_i}$, inclusive. The routes are given IDs from 1 to **M** in the same order of the input.

This is followed by a line consisting of **P** unique integers in the range from 1 to **M**. These represent, in order, the shuttle routes I am taking you on. Each one is an ID of a route from the previous list.

Output

For each test case, output one line containing "Case #x: n", where x is the case number (starting from 1) and n is the ID of the first shuttle route in my path that could not possibly be part of the shortest path from Mountain View to London. If there is no such route, print "Looks Good To Me" instead.

Limits

$1 \le \mathbf{T} \le 10$.
$1 \le \mathbf{u_i}, \mathbf{v_i} \le N$.
$1 \le \mathbf{a_i} \le \mathbf{b_i} \le 1000000$.

My path is guaranteed to be a valid path from Mountain View (city #1) to London (city #2).

There might be more than one shuttle route between the same two cities, and there might be a shuttle route going from a city to itself. Also the suggested path might visit the same city more than once, but it will not use the same shuttle route more than once.

Small dataset

2 ≤ **N** ≤ 20.
1 ≤ **M** ≤ 20.
1 ≤ **P** ≤ 10

Large dataset

2 ≤ **N** ≤ 1000.
1 ≤ **M** ≤ 2000.
1 ≤ **P** ≤ 500.

Sample

Input	Output
3 4 5 3 1 2 100 1000 1 3 500 5000 3 2 400 600 3 4 500 5000 4 2 1 10000 2 4 5 3 3 2 1 3 1 1 3 2 1 1 1 2 1 2 1 2 5 6 3 1 3 1 1 4 2 1 9 1 4 1 1 3 5 2 2 5 2 2 2 3 4 1 2 1 6 2	Case #1: 4 Case #2: Looks Good To Me Case #3: 6

Problem D. Observation Wheel

Small input 8 points	Solve D-small
Large input 23 points	Solve D-large

Problem

An *observation wheel* consists of **N** passenger *gondolas* arranged in a circle, which is slowly rotating. Gondolas pass the entrance one by one, and when a gondola passes the entrance, a person may enter that gondola.

In this problem, the gondolas are so small that they can take just one person each, so if the gondola passing by the entrance is already occupied, the person waiting at the entrance will have to wait for the next one to arrive. If that gondola is also occupied, the person will have to wait for the next one after that, and so on, until a free gondola arrives. For simplicity, we will not consider people exiting the gondolas in this problem — let's assume that all people do is enter the gondolas, and then rotate with the wheel for an arbitrarily long time.

We want to make sure people are not disappointed because of long waiting times, and so we have introduced a flexible pricing scheme: when a person approaches the wheel, and the first gondola passing by the entrance is free, she pays **N** dollars for the ride. If the first gondola is occupied and she has to wait for the second one, she pays **N**-1 dollars for the ride. If the first two gondolas are occupied and she has to wait for the third one, she pays **N**-2 dollars for the ride. Generally, if she has to wait for **K** occupied gondolas to pass by, she pays **N**-**K** dollars. In the worst case, when she has to wait for all but one gondola to pass, she will pay just 1 dollar.

Let's assume that people approach our wheel at random moments in time, so for each person approaching the wheel, the first gondola to pass the entrance is picked uniformly and independently. Let's also assume that nobody will come to the wheel while there's already at least one person waiting to enter, so that we don't have to deal with queueing. A person will always take the first free gondola that passes the entrance.

You are given the number of gondolas and which gondolas are already occupied. How much money are we going to make, on average, until all gondolas become occupied?

Input

The first line of the input gives the number of test cases, **T**. **T** lines follow. Each line describes one test case and contains only '.' (dot) or 'X' (capital letter X) characters. The number of characters in this line gives you **N**. The **i**-th character is 'X' when the **i**-th gondola is already occupied, and '.' when it's still free. The gondolas are numbered in the order they pass the entrance, so the 1st gondola is followed by the 2nd gondola, and so on, starting over from the beginning after the last gondola passes.

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 average amount of money we will get, in dollars. Answers with absolute or relative error no larger than 10^{-9} will be accepted. See the FAQ for an explanation of what that means, and what formats of floating-point numbers we accept.

Limits

$1 \le \mathbf{T} \le 50$.

Small dataset

$1 \le \mathbf{N} \le 20$.

Large dataset

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

Sample

Input	Output
5 .X. X.X.	Case #1: 4.66666666666667 Case #2: 6.00000000000000 Case #3: 5.75000000000000

```
.XX.     Case #4: 13.4722222222222
X..XX.   Case #5: 13.5277777777778
.XX..X
```

Notes

Here's how the first example works. There are nine possibilities, each with probability 1/9:

The first person comes. If the next gondola to pass the entrance is:

- The 1st gondola, which is free, the first person enters it and pays 3 dollars. Then, some time later, the second person comes. If the next gondola to pass the entrance is:
  - The 1st gondola, which is occupied, and so is the 2nd gondola, the second person has to wait until the 3rd gondola, and thus she pays just 1 dollar before entering it. In total, we've earned 4 dollars.
  - The 2nd gondola, which is occupied, the second person has to skip it and enter the 3rd gondola and thus pays 2 dollars. In total, we've earned 5 dollars.
  - The 3rd gondola, which is free, so the second person pays 3 dollars. In total, we've earned 6 dollars.
- The 2nd gondola, which is occupied, the first person has to skip it and enter the 3rd gondola, paying 2 dollars. Then, some time later, the second person comes. If the next gondola to pass the entrance is:
  - The 1st gondola, which is free, the second person pays 3 dollars. In total, we've earned 5 dollars.
  - The 2nd gondola, which is occupied (as is the 3rd gondola), the second person has to wait until the 1st gondola, and thus she pays just 1 dollar before entering it. In total, we've earned 3 dollars.
  - The 3rd gondola, which is occupied, the second person has to skip it and enter the 1st gondola and thus pays 2 dollars. In total, we've earned 4 dollars.
- The 3rd gondola, which is free, the first person enters it and pays 3 dollars. Then, some time later, the second person comes. If the next gondola to pass the entrance is:
  - The 1st gondola, which is free, the second person pays 3 dollars. In total, we've earned 6 dollars.
  - The 2nd gondola, which is occupied (as is the 3rd gondola), the second person has to wait until the 1st gondola, and thus she pays just 1 dollar before entering it. In total, we've earned 4 dollars.
  - The 3rd gondola, which is occupied, the second person has to skip it and enter the 1st gondola and thus pays 2 dollars. In total, we've earned 5 dollars.

We have nine possibilities, earning 3 dollars in one of them, 4 dollars in three of them, 5 dollars in three of them, and 6 dollars in two of them. On average, we earn (1*3+3*4+3*5+2*6)/9=42/9=4.6666666666... dollars.

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

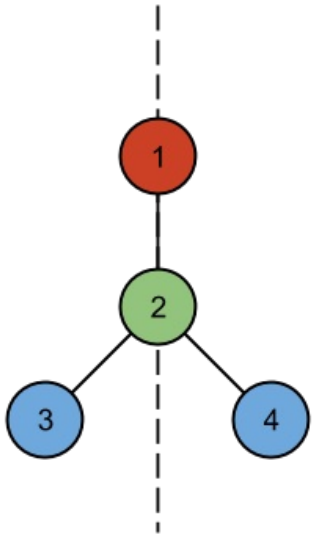
$2 \le \mathbf{N} \le 10000$.

Sample

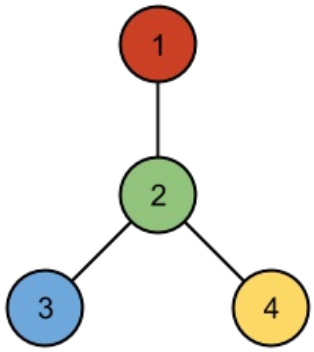
Input	Output
3 4 R G B B 1 2 2 3 2 4 4 R G B Y 1 2 2 3 2 4 12	Case #1: SYMMETRIC Case #2: NOT SYMMETRIC Case #3: SYMMETRIC

```
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
```

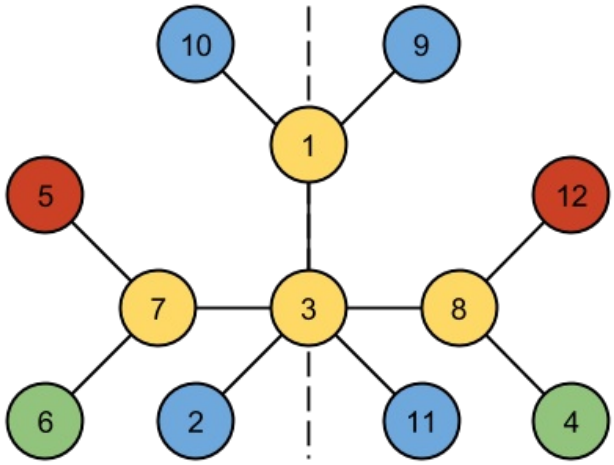
The first case can be drawn as follows:



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.