Problem A. Standing Ovation

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

Problem

It's opening night at the opera, and your friend is the prima donna (the lead female singer). You will not be in the audience, but you want to make sure she receives a standing ovation -- with every audience member standing up and clapping their hands for her.

Initially, the entire audience is seated. Everyone in the audience has a *shyness level*. An audience member with shyness level S_i will wait until at least S_i other audience members have already stood up to clap, and if so, she will immediately stand up and clap. If $S_i = 0$, then the audience member will always stand up and clap immediately, regardless of what anyone else does. For example, an audience member with $S_i = 2$ will be seated at the beginning, but will stand up to clap later after she sees at least two other people standing and clapping.

You know the shyness level of everyone in the audience, and you are prepared to invite additional friends of the prima donna to be in the audience to ensure that everyone in the crowd stands up and claps in the end. Each of these friends may have any shyness value that you wish, not necessarily the same. What is the minimum number of friends that you need to invite to guarantee a standing ovation?

Input

The first line of the input gives the number of test cases, **T**. **T** test cases follow. Each consists of one line with S_{max}, the maximum shyness level of the shyest person in the audience, followed by a string of $S_{max} + 1$ single digits. The **k**th digit of this string (counting starting from 0) represents how many people in the audience have shyness level **k**. For example, the string "409" would mean that there were four audience members with $S_i = 0$ and nine audience members with $S_i = 2$ (and none with $S_i = 1$ or any other value). Note that there will initially always be between 0 and 9 people with each shyness level.

The string will never end in a 0. Note that this implies that there will always be at least one person in the audience.

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 friends you must invite.

Limits

$1 \leq T \leq 100$.

Small dataset

$0 \leq S_{max} \leq 6$.

Large dataset

$0 \leq S_{max} \leq 1000$.

Sample

Input	Output
4 4 11111 1 09 5 110011 0 1	Case #1: 0 Case #2: 1 Case #3: 2 Case #4: 0

In Case #1, the audience will eventually produce a standing ovation on its own,

without you needing to add anyone -- first the audience member with $S_i = 0$ will stand up, then the audience member with $S_i = 1$ will stand up, etc.

In Case #2, a friend with $S_i = 0$ must be invited, but that is enough to get the entire audience to stand up.

In Case #3, one optimal solution is to add two audience members with $S_i = 2$.

In Case #4, there is only one audience member and he will stand up immediately. No friends need to be invited.

Problem B. Infinite House of Pancakes

Small input 9 points	Solve B-small
Large input 12 points	Solve B-large

Problem

At the Infinite House of Pancakes, there are only finitely many pancakes, but there are infinitely many diners who would be willing to eat them! When the restaurant opens for breakfast, among the infinitely many diners, exactly **D** have non-empty plates; the **i**th of these has $\mathbf{P_i}$ pancakes on his or her plate. Everyone else has an empty plate.

Normally, every minute, every diner with a non-empty plate will eat one pancake from his or her plate. However, some minutes may be *special*. In a special minute, the head server asks for the diners' attention, chooses a diner with a non-empty plate, and carefully lifts some number of pancakes off of that diner's plate and moves those pancakes onto one other diner's (empty or non-empty) plate. No diners eat during a special minute, because it would be rude.

You are the head server on duty this morning, and it is your job to decide which minutes, if any, will be special, and which pancakes will move where. That is, every minute, you can decide to either do nothing and let the diners eat, or declare a special minute and interrupt the diners to make a single movement of one or more pancakes, as described above.

Breakfast ends when there are no more pancakes left to eat. How quickly can you make that happen?

Input

The first line of the input gives the number of test cases, **T**. **T** test cases follow. Each consists of one line with **D**, the number of diners with non-empty plates, followed by another line with **D** space-separated integers representing the numbers of pancakes on those diners' plates.

Output

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

Limits

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

Small dataset

$1 \leq \mathbf{D} \leq 6$.
$1 \leq \mathbf{P_i} \leq 9$.

Large dataset

$1 \leq \mathbf{D} \leq 1000$.
$1 \leq \mathbf{P_i} \leq 1000$.

Sample

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

In Case #1, one diner starts with 3 pancakes and everyone else's plate is empty. One optimal strategy is:

Minute 1: Do nothing. The diner will eat one pancake.

Minute 2 (special): Interrupt and move one pancake from that diner's stack onto another diner's empty plate. (Remember that there are always infinitely many diners with empty plates available, no matter how many diners start off with pancakes.) No pancakes are eaten during an interruption.

Minute 3: Do nothing. Each of those two diners will eat one of the last two remaining pancakes.

In Case #2, it is optimal to let the diners eat for 2 minutes, with no interruptions, during which time they will finish all the pancakes.

In Case #3, one diner starts with 4 pancakes and everyone else's plate is empty. It is optimal to use the first minute as a special minute to move two pancakes from the diner's plate to another diner's empty plate, and then do nothing and let the diners eat for the second and third minutes.

Problem C. Dijkstra

Small input 11 points	Solve C-small
Large input 17 points	Solve C-large

Problem

The Dutch computer scientist Edsger Dijkstra made many important contributions to the field, including the shortest path finding algorithm that bears his name. This problem is not about that algorithm.

You were marked down one point on an algorithms exam for misspelling "Dijkstra" -- between `D` and `stra`, you wrote some number of characters, each of which was either `i`, `j`, or `k`. You are prepared to argue to get your point back using *quaternions*, an actual number system (extended from complex numbers) with the following multiplicative structure:

	1	***i***	***j***	***k***
1	1	*i*	*j*	*k*
i	*i*	-1	*k*	*-j*
j	*j*	*-k*	-1	*i*
k	*k*	*j*	*-i*	-1

To multiply one quaternion by another, look at the row for the first quaternion and the column for the second quaternion. For example, to multiply *i* by *j*, look in the row for *i* and the column for *j* to find that the answer is *k*. To multiply *j* by *i*, look in the row for *j* and the column for *i* to find that the answer is *-k*.

As you can see from the above examples, the quaternions are not commutative -- that is, there are some **a** and **b** for which **a** * **b** != **b** * **a**. However they are associative -- for any **a**, **b**, and **c**, it's true that **a** * (**b** * **c**) = (**a** * **b**) * **c**.

Negative signs before quaternions work as they normally do -- for any quaternions **a** and **b**, it's true that **-a** * **-b** = **a** * **b**, and **-a** * **b** = **a** * **-b** = **-(a** * **b)**.

You want to argue that your misspelling was equivalent to the correct spelling `ijk` by showing that you can split your string of `i`s, `j`s, and `k`s in two places, forming three substrings, such that the leftmost substring reduces (under quaternion multiplication) to *i*, the middle substring reduces to *j*, and the right substring reduces to *k*. (For example, `jij` would be interpreted as *j* * *i* * *j*; *j* * *i* is *-k*, and *-k* * *j* is *i*, so `jij` reduces to *i*.) If this is possible, you will get your point back. Can you find a way to do it?

Input

The first line of the input gives the number of test cases, **T**. **T** test cases follow. Each consists of one line with two space-separated integers **L** and **X**, followed by another line with **L** characters, all of which are `i`, `j`, or `k`. Note that the string never contains negative signs, 1s, or any other characters. The string that you are to evaluate is the given string of **L** characters repeated **X** times. For instance, for **L** = 4, **X** = 3, and the given string `kiij`, your input string would be `kiijkiijkiij`.

Output

For each test case, output one line containing "Case #x: y", where x is the test case number (starting from 1) and y is either `YES` or `NO`, depending on whether the string can be broken into three parts that reduce to *i*, *j*, and *k*, in that order, as described above.

Limits

1 ≤ **T** ≤ 100.
1 ≤ **L** ≤ 10000.

Small dataset

1 ≤ **X** ≤ 10000.
1 ≤ **L** * **X** ≤ 10000.

Large dataset

$1 \le \mathbf{X} \le 10^{12}$.
$1 \le \mathbf{L} * \mathbf{X} \le 10^{16}$.

Sample

Input	Output
5 2 1 ik 3 1 ijk 3 1 kji 2 6 ji 1 10000 i	Case #1: NO Case #2: YES Case #3: NO Case #4: YES Case #5: NO

In Case #1, the string is too short to be split into three substrings.

In Case #2, just split the string into `i`, `j`, and `k`.

In Case #3, the only way to split the string into three parts is `k`, `j`, `i`, and this does not satisfy the conditions.

In Case #4, the string is `jijijijijii`. It can be split into `jij` (which reduces to *i*), `iji` (which reduces to *j*), and `jijiji` (which reduces to *k*).

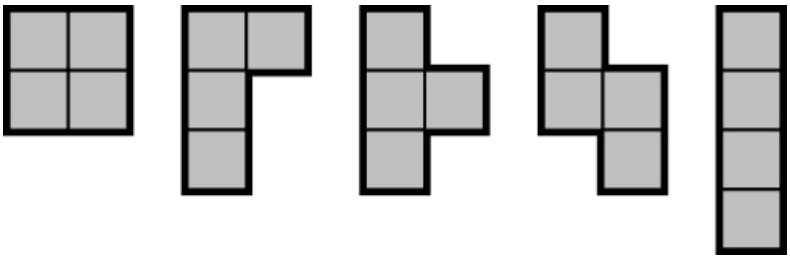
In Case #5, no matter how you choose your substrings, none of them can ever reduce to a *j* or a *k*.

Problem D. Ominous Omino

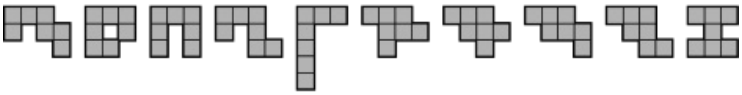
Small input 8 points	Solve D-small
Large input 26 points	Solve D-large

Problem

An *N*-omino is a two-dimensional shape formed by joining *N* unit cells fully along their edges in some way. More formally, a 1-omino is a 1x1 unit square, and an *N*-omino is an (*N*-1)omino with one or more of its edges joined to an adjacent 1x1 unit square. For the purpose of this problem, we consider two *N*-ominoes to be the same if one can be transformed into the other via reflection and/or rotation. For example, these are the five possible 4-ominoes:



And here are some of the 108 possible 7-ominoes:



Richard and Gabriel are going to play a game with the following rules, for some predetermined values of **X**, **R**, and **C**:

1. Richard will choose any one of the possible **X**-ominoes.
2. Gabriel must use at least one copy of that **X**-omino, along with arbitrarily many copies of any **X**-ominoes (which can include the one Richard chose), to completely fill in an **R**-by-**C** grid, with no overlaps and no spillover. That is, every cell must be covered by exactly one of the **X** cells making up an **X**-omino, and no **X**-omino can extend outside the grid. Gabriel is allowed to rotate or reflect as many of the **X**-ominoes as he wants, including the one Richard chose. If Gabriel can completely fill in the grid, he wins; otherwise, Richard wins.

Given particular values **X**, **R**, and **C**, can Richard choose an **X**-omino that will ensure that he wins, or is Gabriel guaranteed to win no matter what Richard chooses?

Input

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

Output

For each test case, output one line containing "Case #x: y", where x is the test case number (starting from 1) and y is either RICHARD (if there is at least one choice that ensures victory for Richard) or GABRIEL (if Gabriel will win no matter what Richard chooses).

Limits

Small dataset

T = 64.
1 ≤ **X, R, C** ≤ 4.

Large dataset

1 ≤ **T** ≤ 100.
1 ≤ **X, R, C** ≤ 20.

Sample

Input	Output
4 2 2 2 2 1 3 4 4 1 3 2 3	Case #1: GABRIEL Case #2: RICHARD Case #3: RICHARD Case #4: GABRIEL

In case #1, Richard only has one 2-omino available to choose -- the 1x2 block formed by joining two unit cells together. No matter how Gabriel places this block in the 2x2 grid, he will leave a hole that can be exactly filled with another 1x2 block. So Gabriel wins.

In case #2, Richard has to choose the 1x2 block, but no matter where Gabriel puts it, he will be left with a single 1x1 hole that he cannot fill using only 2-ominoes. So Richard wins.

In case #3, one winning strategy for Richard is to choose the 2x2 square 4-omino. There is no way for Gabriel to fit that square into the 4x1 grid such that it is completely contained within the grid, so Richard wins.

In case #4, Richard can either pick the straight 3-omino or the L-shaped 3-omino. In either case, Gabriel can fit it into the grid and then use another copy of the same 3-omino to fill in the remaining hole.

Round 1A 2015

A. Mushroom Monster
B. Haircut
C. Logging

Contest Analysis
Questions asked

Problem A. Mushroom Monster

Small input 7 points	Solve A-small
Large input 8 points	Solve A-large

Problem

Kaylin loves mushrooms. Put them on her plate and she'll eat them up! In this problem she's eating a plate of mushrooms, and Bartholomew is putting more pieces on her plate.

In this problem, we'll look at how many pieces of mushroom are on her plate at 10-second intervals. Bartholomew could put any non-negative integer number of mushroom pieces down at any time, and the only way they can leave the plate is by being eaten.

Figure out the minimum number of mushrooms that Kaylin could have eaten using two different methods of computation:

1. Assume Kaylin could eat any number of mushroom pieces at any time.
2. Assume that, starting with the first time we look at the plate, Kaylin eats mushrooms at a constant rate whenever there are mushrooms on her plate.

For example, if the input is `10 5 15 5`:

With the first method, Kaylin must have eaten at least 15 mushroom pieces: first she eats 5, then 10 more are put on her plate, then she eats another 10. There's no way she could have eaten fewer pieces.

With the second method, Kaylin must have eaten at least 25 mushroom pieces. We can determine that she must eat mushrooms at a rate of at least 1 piece per second. She starts with 10 pieces on her plate. In the first 10 seconds, she eats 10 pieces, and 5 more are put on her plate. In the next 5 seconds, she eats 5 pieces, then her plate stays empty for 5 seconds, and then Bartholomew puts 15 more pieces on her plate. Then she eats 10 pieces in the last 10 seconds.

Input

The first line of the input gives the number of test cases, **T**. **T** test cases follow. Each will consist of one line containing a single integer **N**, followed by a line containing **N** space-separated integers $\mathbf{m}_i$; the number of mushrooms on Kaylin's plate at the start, and at 10-second intervals.

Output

For each test case, output one line containing "Case #x: y z", where x is the test case number (starting from 1), y is the minimum number of mushrooms Kaylin could have eaten using the first method of computation, and z is the minimum number of mushrooms Kaylin could have eaten using the second method of computation.

Limits

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

Small dataset

$2 \leq \mathbf{N} \leq 10$.
$0 \leq \mathbf{m}_i \leq 100$.

Large dataset

$2 \leq \mathbf{N} \leq 1000$.
$0 \leq \mathbf{m}_i \leq 10000$.

Sample

Input	Output
4 4 10 5 15 5 2 100 100	Case #1: 15 25 Case #2: 0 0 Case #3: 81 567 Case #4: 181 244

```
8
81 81 81 81 81 81 81 0
6
23 90 40 0 100 9
```

Round 1A 2015

A. Mushroom Monster
B. Haircut
C. Logging

Contest Analysis
Questions asked

Problem B. Haircut

Small input
11 points

Solve B-small

Large input
22 points

Solve B-large

Problem

You are waiting in a long line to get a haircut at a trendy barber shop. The shop has **B** barbers on duty, and they are numbered 1 through **B**. It always takes the **k**th barber exactly $\mathbf{M_k}$ minutes to cut a customer's hair, and a barber can only cut one customer's hair at a time. Once a barber finishes cutting hair, he is immediately free to help another customer.

While the shop is open, the customer at the head of the queue always goes to the lowest-numbered barber who is available. When no barber is available, that customer waits until at least one becomes available.

You are the **N**th person in line, and the shop has just opened. Which barber will cut your hair?

Input

The first line of the input gives the number of test cases, **T**. **T** test cases follow; each consists of two lines. The first contains two space-separated integers **B** and **N** -- the number of barbers and your place in line. The customer at the head of the line is number 1, the next one is number 2, and so on. The second line contains $\mathbf{M_1}$, $\mathbf{M_2}$, ..., $\mathbf{M_B}$.

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 the barber who will cut your hair.

Limits

$1 \le \mathbf{T} \le 100$.
$1 \le \mathbf{N} \le 10^9$.

Small dataset

$1 \le \mathbf{B} \le 5$.
$1 \le \mathbf{M_k} \le 25$.

Large dataset

$1 \le \mathbf{B} \le 1000$.
$1 \le \mathbf{M_k} \le 100000$.

Sample

Input	Output
3 2 4 10 5 3 12 7 7 7 3 8 4 2 1	Case #1: 1 Case #2: 3 Case #3: 1

In Case #1, you are the fourth person in line, and barbers 1 and 2 take 10 and 5 minutes, respectively, to cut hair. When the shop opens, the first customer immediately has the choice of barbers 1 and 2, and she will choose the lowest-numbered barber, 1. The second customer will immediately be served by barber 2. The third customer will wait since there are no more free barbers. After 5 minutes, barber 2 will finish cutting the second customer's hair, and will serve the third customer. After 10 minutes, both barbers 1 and 2 will finish; you are next in line, and you will have the choice of barbers 1 and 2, and will choose 1.

Round 1A 2015

A. Mushroom Monster
B. Haircut
C. Logging

Contest Analysis
Questions asked

Problem C. Logging

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 18 points	Solve C-small
Large input 34 points	Solve C-large

Problem

A certain forest consists of **N** trees, each of which is inhabited by a squirrel.

The **boundary** of the forest is the convex polygon of smallest area which contains every tree, as if a giant rubber band had been stretched around the outside of the forest.

Formally, every tree is a single point in two-dimensional space with unique coordinates ($\mathbf{X_i}$, $\mathbf{Y_i}$), and the boundary is the convex hull of those points.

Some trees are **on the boundary** of the forest, which means they are on an edge or a corner of the polygon. The squirrels wonder how close their trees are to being on the boundary of the forest.

One at a time, each squirrel climbs down from its tree, examines the forest, and determines the minimum number of trees that would need to be cut down for its own tree to be on the boundary. It then writes that number down on a log.

Determine the list of numbers written on the log.

Input

The first line of the input gives the number of test cases, **T**. **T** test cases follow; each consists of a single line with an integer **N**, the number of trees, followed by **N** lines with two space-separated integers $\mathbf{X_i}$ and $\mathbf{Y_i}$, the coordinates of each tree. No two trees will have the same coordinates.

Output

For each test case, output one line containing "Case #x:", followed by **N** lines with one integer each, where line **i** contains the number of trees that the squirrel living in tree **i** would need to cut down.

Limits

$-10^6 \leq \mathbf{X_i}, \mathbf{Y_i} \leq 10^6$.

Small dataset

$1 \leq \mathbf{T} \leq 100$.
$1 \leq \mathbf{N} \leq 15$.

Large dataset

$1 \leq \mathbf{T} \leq 14$.
$1 \leq \mathbf{N} \leq 3000$.

Sample

Input	Output
2 5 0 0 10 0 10 10 0 10 5 5 9 0 0 5 0 10 0 0 5 5 5 10 5 0 10 5 10 10 10	Case #1: 0 0 0 0 1 Case #2: 0 0 0 0 3 0 0 0 0

In the first sample case, there are four trees forming a square, and a fifth tree inside the square. Since the first four trees are already on the boundary, the squirrels for those trees each write down 0. Since one tree needs to be cut down for the fifth tree to be on the boundary, the fifth squirrel writes down 1.

Problem A. Counter Culture

Small input 11 points	Solve A-small
Large input 14 points	Solve A-large

Problem

In the Counting Poetry Slam, a performer takes the microphone, chooses a number **N**, and counts aloud from 1 to **N**. That is, she starts by saying 1, and then repeatedly says the number that is 1 greater than the previous number she said, stopping after she has said **N**.

It's your turn to perform, but you find this process tedious, and you want to add a twist to speed it up: sometimes, instead of adding 1 to the previous number, you might reverse the digits of the number (removing any leading zeroes that this creates). For example, after saying "16", you could next say either "17" or "61"; after saying "2300", you could next say either "2301" or "32". You may reverse as many times as you want (or not at all) within a performance.

The first number you say must be 1; what is the fewest number of numbers you will need to say in order to reach the number **N**? 1 and **N** count toward this total. If you say the same number multiple times, each of those times counts separately.

Input

The first line of the input gives the number of test cases, **T**. **T** lines follow. Each has one integer **N**, the number you must reach.

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 numbers you need to say.

Limits

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

Small dataset

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

Large dataset

$1 \leq \mathbf{N} \leq 10^{14}$.

Sample

Input	Output
3 1 19 23	Case #1: 1 Case #2: 19 Case #3: 15

In Case #2, flipping does not help and the optimal strategy is to just count up to 19.

In Case #3, the optimal strategy is to count up to 12, flip to 21, and then continue counting up to 23. That is, the numbers you will say are 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 21, 22, 23.


Powered by
Google Cloud Platform

Problem B. Noisy Neighbors

Small input 12 points	Solve B-small
Large input 15 points	Solve B-large

Problem

You are a landlord who owns a building that is an **R** x **C** grid of apartments; each apartment is a unit square cell with four walls. You want to rent out **N** of these apartments to tenants, with exactly one tenant per apartment, and leave the others empty. Unfortunately, all of your potential tenants are noisy, so whenever any two occupied apartments share a wall (and not just a corner), this will add one point of *unhappiness* to the building. For example, a 2x2 building in which every apartment is occupied has four walls that are shared by neighboring tenants, and so the building's unhappiness score is 4.

If you place your **N** tenants optimally, what is the minimum unhappiness value for your building?

Input

The first line of the input gives the number of test cases, **T**. **T** lines follow; each contains three space-separated integers: **R**, **C**, and **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 minimum possible unhappiness for the building.

Limits

$1 \le \mathbf{T} \le 1000$.
$0 \le \mathbf{N} \le \mathbf{R}*\mathbf{C}$.

Small dataset

$1 \le \mathbf{R}*\mathbf{C} \le 16$.

Large dataset

$1 \le \mathbf{R}*\mathbf{C} \le 10000$.

Sample

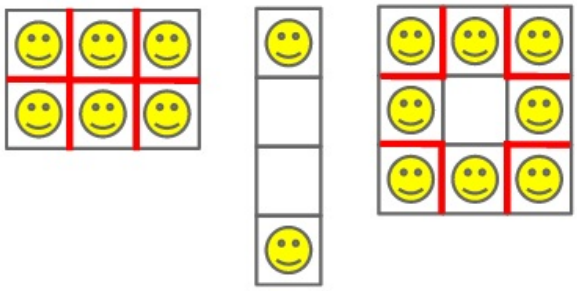
Input	Output
4 2 3 6 4 1 2 3 3 8 5 2 0	Case #1: 7 Case #2: 0 Case #3: 8 Case #4: 0

In Case #1, every room is occupied by a tenant and all seven internal walls have tenants on either side.

In Case #2, there are various ways to place the two tenants so that they do not share a wall. One is illustrated below.

In Case #3, the optimal strategy is to place the eight tenants in a ring, leaving the middle apartment unoccupied.

Here are illustrations of sample cases 1-3. Each red wall adds a point of unhappiness.



Problem C. Hiking Deer

Small input 1 13 points	Solve C-small-1
Small input 2 16 points	Solve C-small-2
Large input 19 points	Solve C-large

Problem

Herbert Hooves the deer is going for a hike: one clockwise loop around his favorite circular trail, starting at degree zero. Herbert has perfect control over his speed, which can be *any* nonnegative value (not necessarily an integer) at any time -- he can change his speed instantaneously whenever he wants. When Herbert reaches his starting point again, the hike is over.

The trail is also used by human hikers, who also walk clockwise around the trail. Each hiker has a starting point and moves at her own constant speed. Humans continue to walk around and around the trail forever.

Herbert is a skittish deer who is afraid of people. He does not like to have encounters with hikers. An encounter occurs whenever Herbert and a hiker are in exactly the same place at the same time. You should consider Herbert and the hikers to be points on the circumference of a circle.

Herbert can have multiple separate encounters with the same hiker.

If more than one hiker is encountered at the same instant, all of them count as separate encounters.

Any encounter at the exact instant that Herbert finishes his hike still counts as an encounter.

If Herbert were to have an encounter with a hiker and then change his speed to exactly match that hiker's speed and follow along, he would have infinitely many encounters! Of course, he must never do this.

Encounters do not change the hikers' behavior, and nothing happens when hikers encounter each other.

Herbert knows the starting position and speed of each hiker. What is the minimum number of encounters with hikers that he can possibly have?

Solving this problem

Usually, Google Code Jam problems have 1 Small input and 1 Large input. This problem has 2 Small inputs and 1 Large input. You must solve the first Small input before you can attempt the second Small input; as usual, you will be able to retry the Small inputs (with a time penalty). Once you have solved both Small inputs, you will be able to download the Large input; as usual, you will get only one chance at the Large input.

Input

The first line of the input gives the number of test cases, **T**. **T** test cases follow. Each begins with one line with an integer **N**, and is followed by **N** lines, each of which represents a *group* of hikers starting at the same position on the trail. The **i**th of these lines has three space-separated integers: a starting position $\mathbf{D_i}$ (representing $\mathbf{D_i}$/360ths of the way around the trail from the deer's starting point), the number $\mathbf{H_i}$ of hikers in the group, and $\mathbf{M_i}$, the amount of time (in minutes) it takes for the fastest hiker in that group to make each complete revolution around the circle. The other hikers in that group each complete a revolution in $\mathbf{M_i}$+1, $\mathbf{M_i}$+2, ..., $\mathbf{M_i}$+$\mathbf{H_i}$-1 minutes. For example, the line

```
180 3 4
```

would mean that three hikers begin halfway around the trail from the deer's starting point, and that they take 4, 5, and 6 minutes, respectively, to complete each full revolution around the trail.

Herbert always starts at position 0 (0/360ths of the way around the circle), and no group of hikers does. Multiple groups of hikers may begin in the same place, but no two hikers will both begin in the same place and have the same speed.

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 encounters with hikers that the deer can have.

Limits

$1 \le \mathbf{T} \le 100$.
$1 \le \mathbf{D_i} \le 359$.
$1 \le \mathbf{N} \le 1000$.
$1 \le \mathbf{H_i}$.
$1 \le \mathbf{M_i} \le 10^9$. (Note that this only puts a cap on the time required for the *fastest* hiker in each group to complete a revolution. Slower hikers in the group will take longer.)

Small dataset 1

The total number of hikers in each test case will not exceed 2.

Small dataset 2

The total number of hikers in each test case will not exceed 10.

Large dataset

The total number of hikers in each test case will not exceed 500000.

Sample

Input	Output
3 4 1 1 12 359 1 12 2 1 12 358 1 12 2 180 1 100000 180 1 1 1 180 2 1	Case #1: 0 Case #2: 1 Case #3: 0

In Case #1, the hikers all happen to be moving at the same speed, and one way for Herbert to avoid encountering any of them is to move exactly as fast as they do.

In Case #2, the second hiker is moving much faster than the first, and if Herbert goes slowly enough to avoid overtaking the first hiker, he will have multiple encounters with the speedy second hiker. One optimal strategy for Herbert is to go exactly as fast as the second hiker, encountering the first hiker once and never encountering the second hiker at all.

In Case #3, the two hikers start in the same place, but one is twice as fast as the other. One optimal strategy is for Herbert to immediately catch up to the slower hiker without overtaking him, follow just behind him until he passes the deer's starting position, and then finish quickly before the faster hiker can catch Herbert.

Problem A. Brattleship

Small input
11 points
Solve A-small

Large input
22 points
Solve A-large

Problem

You're about to play a simplified "battleship" game with your little brother. The board for this game is a rectangular grid with **R** rows and **C** columns. At the start of the game, you will close your eyes, and you will keep them closed until the end of the game. Your little brother will take a single rectangular 1 x **W** ship and place it **horizontally** somewhere on the board. The ship must always fit entirely on the board, with each cell of the ship occupying exactly one of the grid's cells, and it can never be rotated.

In each turn of the game, you name a cell on the board, and your little brother tells you whether that is a hit (one of the cells occupied by the ship) or a miss. (Your little brother doesn't say *which* part of the ship was hit -- just that the cell you named has a part of the ship in it.) You have perfect memory, and can keep track of all the information he has given you. Once you have named all of the cells occupied by the ship, the game is over (the ship is sunk), and your score is the number of turns taken. Your goal is to minimize your score.

Although the ship is not supposed to be moved once it is placed, you know that your little brother, who is a brat, plans to cheat by changing the location of the ship whenever he wants, as long as the ship remains horizontal and completely on the board, and the new location is consistent with all the information he has given so far. For example, for a 1x4 board and 1x2 ship, your little brother could initially place the ship such that it overlaps the leftmost two columns. If your first guess was row 1, column 2, he could choose to secretly move the ship to the rightmost two columns, and tell you that (1, 2) was a miss. If your next guess after that was (1, 3), though, then he could not say that was also a miss and move the ship back to its original location, since that would be inconsistent with what he said about (1, 2) earlier.

Not only do you know that your little brother will cheat, he knows that you know. If you both play optimally (you to minimize your score, him to maximize it), what is the lowest score that you can **guarantee** you will achieve, regardless of what your little brother does?

Input

The first line of the input gives the number of test cases, **T**. **T** lines follow, each with three space-separated integers **R**, **C**, and **W**: the number of rows and columns of the board, followed by the width of the ship.

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 score you can guarantee.

Limits

$1 \leq \mathbf{W} \leq \mathbf{C}$.

Small dataset

T = 55.
R = 1.
$1 \leq \mathbf{C} \leq 10$.

Large dataset

$1 \leq \mathbf{T} \leq 100$.
$1 \leq \mathbf{R} \leq 20$.
$1 \leq \mathbf{C} \leq 20$.

Sample

Input	Output
3 1 4 2 1 7 7 2 5 1	Case #1: 3 Case #2: 7 Case #3: 10

In Case #1, the board has one row and four columns, and the ship takes up one row and two columns. One optimal strategy is for you to start by naming cell (1, 2):

If your little brother says it is a hit, then the other cell of the 1x2 ship must be in either (1, 1) or (1, 3), and you just have to name both. If you happen to correctly name the cell where the other part of the ship is, your little brother will just reposition the ship so that (1, 2) is still hit, but your guess is a miss. Notice that your little brother can still move the ship even after it has been hit, as long as the new position is not inconsistent with the information he has already given.

If your little brother says it is a miss, then the only remaining consistent scenario is that the ship is in (1, 3) and (1, 4), and your little brother will be unable to change this from now on; you just need to name those two cells.

So no matter what your little brother does after you say (1, 2), you can finish the game in two more moves after that, for a total of three moves.

Moreover, a three-move solution is optimal, because it is impossible to guarantee a finish in only two moves: without loss of generality, pick a first move. No matter what you pick, there is still a 1x2 area open and your little brother can just move the ship there and claim that you missed. It is impossible for you to sink that ship, which has not yet been hit, with only one more move.

In Case #2, the ship completely fills in the board and so your little brother has only one place to put it. All you have to do is name every cell.

In Case #3, your little brother can always move the 1x1 ship to a cell you have not tried yet, so you must name all 10 cells, only finally getting a hit (and immediately sinking the ship) on the last one.

Problem B. Typewriter Monkey

Small input 11 points	Solve B-small
Large input 22 points	Solve B-large

Problem

Your publishing house has decided to use monkeys randomly typing at keyboards to write great works of literature. You are the supervisor for one monkey with a keyboard containing **K** keys, each of which is labeled with an uppercase English letter. (There may be multiple keys displaying the same letter.) The monkey will start with an empty string and repeat the following **S** times: choose a key from its keyboard uniformly at random and press it, adding a copy of that key's letter to the right end of the string. The final resulting string will have length **S**.

You have a *target word* of length **L** that you are hoping the monkey will type. (The target word will not necessarily be a real English word.) This target word may even appear multiple times in what the monkey types. (Overlapping instances count too -- for example, if "ABA" is the target word and the monkey types "ABABA", that contains two instances of the target.)

You plan to pay the monkey one banana for each instance of the target word that it types. When you go to inspect the monkey's work, you will bring along the minimum number of bananas that you need to *ensure* that you will always have enough bananas to pay the monkey, no matter what it has typed. Then, you will pay the monkey one banana for each instance of the target word that it *actually* typed. You will keep the remaining bananas that you brought with you.

What is the expected number of bananas that you will get to keep?

Input

The first line of the input gives the number of test cases, **T**. **T** test cases follow. Each consists of three lines. The first contains three space-separated positive integers: **K**, **L**, and **S**. The second contains a string of **K** uppercase English letters representing the monkey's keyboard. The third contains a string of **L** uppercase English letters representing the target word.

Output

For each test case, output one line containing "Case #x: y", where y is the expected number of bananas you will get to keep after paying the monkey.

y will be considered correct if it is within an absolute or relative error of 10^{-6} of the correct answer. See the FAQ for an explanation of what that means, and what formats of real numbers we accept.

Limits

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

Small dataset

$1 \le \mathbf{K} \le 7$.
$1 \le \mathbf{L} \le \mathbf{S} \le 7$.

Large dataset

$1 \le \mathbf{K} \le 100$.
$1 \le \mathbf{L} \le \mathbf{S} \le 100$.

Sample

Input	Output
5 7 6 6 BANANAS MONKEY 2 3 4 AA AAA 2 1 2 AB	Case #1: 0.0 Case #2: 0.0 Case #3: 1.0 Case #4: 0.8888889 Case #5: 9.0

```
B
6 2 2
GOOGLE
GO
26 11 100
ABCDEFGHIJKLMNOPQRSTUVWXYZ
ROSENCRANTZ
```

Note that Case #5 is not within the limits for the Small dataset.

In Case #1, the monkey has no chance of typing the target word "MONKEY" even once (because his keyboard lacks most of the letters in "MONKEY"), so you do not bring any bananas along when you visit, and of course you do not pay any. Poor monkey!

In Case #2, the monkey is guaranteed to type "AAAA", which has two overlapping instances of the target word "AAA". You will bring two bananas and then pay both.

In Case #3, the monkey will produce the following outputs with equal probability (1/4 each): "AA", "AB", "BA", "BB". These have 0, 1, 1, and 2 instances of the target word, respectively. You must bring 2 bananas to be ready for the "BB" case, but you will on average pay (0 + 1 + 1 + 2) / 4 = 1.

In Case #4, the monkey has a 1/3 chance of typing a "G" first and a 1/3 chance of typing an "O" second, for a 1/9 chance of typing "GO". You will bring one banana and give it up 1/9 of the time.

In Case #5, the monkey could in theory type "ROSENCRANTZ" up to nine times, but the chances of this happening even once are so small that they are negligible compared to the acceptable margin of error for answers.

Round 1C 2015

A. Brattleship
B. Typewriter Monkey
C. Less Money, More Problems

Contest Analysis
Questions asked

Problem C. Less Money, More Problems

Small input 11 points	Solve C-small
Large input 23 points	Solve C-large

Problem

Up until today, the nation you live in has used **D** different positive integer denominations of coin for all transactions. Today, the queen got angry when a subject tried to pay his taxes with a giant sack of low-valued coins, and she just decreed that no more than **C** coins of any one denomination may be used in any one purchase. For instance, if **C** = 2 and the existing denominations are 1 and 5, it is possible to buy something of value 11 by using two 5s and one 1, or something of value 12 by using two 5s and two 1s, but it is impossible to buy something of value 9 or 17.

You cannot directly challenge the queen's decree, but you happen to be in charge of the mint, and you *can* issue new denominations of coin. You want to make it possible for *any* item of positive value at most **V** to be purchased under the queen's new rules. (Note that this may not necessarily have been possible before the queen's decree.) Moreover, you want to introduce as few new denominations as possible, and your final combined set of pre-existing and new denominations may not have any repeats.

What is the smallest number of new denominations required?

Input

The first line of the input gives the number of test cases, **T**. **T** test cases follow. Each consists of one line with three space-separated values **C**, **D**, and **V**, followed by another line with **D** distinct space-separated values representing the preexisting denominations, in ascending order.

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 new denominations required, as described above.

Limits

$1 \le \mathbf{T} \le 100$.
Each existing denomination $\le \mathbf{V}$.

Small dataset

$\mathbf{C} = 1$.
$1 \le \mathbf{D} \le 5$.
$1 \le \mathbf{V} \le 30$.

Large dataset

$1 \le \mathbf{C} \le 100$.
$1 \le \mathbf{D} \le 100$.
$1 \le \mathbf{V} \le 10^9$.

Sample

Input	Output
4 1 2 3 1 2 1 3 6 1 2 5 2 1 3 3 1 6 100 1 5 10 25 50 100	Case #1: 0 Case #2: 1 Case #3: 1 Case #4: 3

Note that Cases #3 and #4 are not within the limits for the Small dataset.

In Case #1, it is already possible to make all the required values (1, 2, and 3)

using at most one copy of each of the existing denominations.

In Case #2, it suffices to add a denomination of either 3 or 4 -- whichever you choose, only one new denomination is required.

In Case #3, the optimal solution is to add a denomination of 1.

Problem A. Pegman

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

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

Problem

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

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

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

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

Input

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

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

Output

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

Limits

1 ≤ **T** ≤ 100.

Small dataset

1 ≤ **R, C** ≤ 4.

Large dataset

1 ≤ **R, C** ≤ 100.

Sample

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

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

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

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

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

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

Problem B. Kiddie Pool

Small input
7 points
Solve B-small

Large input
18 points
Solve B-large

Problem

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

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

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

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

Input

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

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

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

Output

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

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

Limits

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

Small dataset

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

Large dataset

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

Sample

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

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

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

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

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

Problem C. Bilingual

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

Problem

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

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

Input

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

Output

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

Limits

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

Small dataset

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

Large dataset

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

Sample

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

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

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

Problem D. Drum Decorator

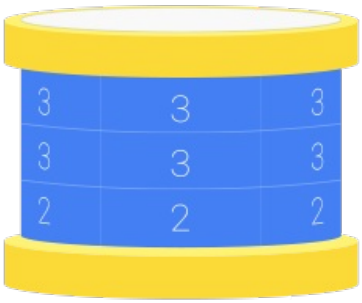
Small input 11 points	Solve D-small
Large input 19 points	Solve D-large

Problem

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

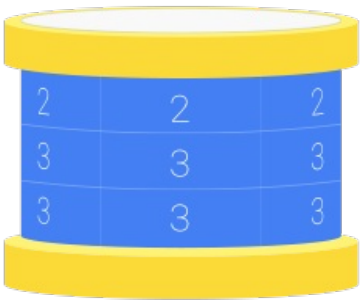
Your band is scheduled to perform in Mathland. The Mathlanders are a tough audience, and they will expect every cell of your drum to contain a positive integer; zeroes and negative integers are not allowed. Moreover, each integer K must border (share an edge, and not just a point, with) exactly K other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

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



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

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



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

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

Input

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

Output

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

Limits

Small dataset

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

Large dataset

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

Sample

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

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

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

Problem A. Fairland

Small input 3 points	Solve A-small
Large input 9 points	Solve A-large

Problem

The country of Fairland has very strict laws governing how companies organize and pay their employees:

1. Each company must have exactly one CEO, who has no manager.
2. Every employee except for the CEO must have exactly one manager. (This means that the org chart showing all of the employees in a company is a tree, without cycles.)
3. As long as an employee is working for the company, their manager must never change. This means that if a manager leaves, then all of the employees reporting to that manager must also leave.
4. The CEO must never leave the company.
5. Every employee receives a salary -- some amount of Fairland dollars per year. An employee's salary must never change.
6. Different employees may have different salaries, and an employee's salary is not necessarily correlated with where in the org chart that employee is.

The government of Fairland has just passed one additional law:

7. The difference between the largest salary and the smallest salary in the whole company must be at most **D** Fairland dollars.

Marie is the CEO of the Fairland General Stuff Corporation, and she has to ensure that her company complies with the new law. This may require laying off some employees. She has the list of the company's employees, their managers, and their salaries. What is the largest number of employees she can keep, including herself?

Input

The first line of the input gives the number of test cases, **T**. **T** test cases follow. Each case begins with one line containing two space-separated integers **N** (the number of employees) and **D** (the maximum allowed salary difference). This is followed by one line with four space-separated integers ($\mathbf{S_0}$, $\mathbf{A_s}$, $\mathbf{C_s}$, $\mathbf{R_s}$) and then another line with four more space-separated integers ($\mathbf{M_0}$, $\mathbf{A_m}$, $\mathbf{C_m}$ and $\mathbf{R_m}$). These last eight integers define the following sequences:

- $\mathbf{S_{i+1}} = (\mathbf{S_i} * \mathbf{A_s} + \mathbf{C_s})$ *mod* $\mathbf{R_s}$
- $\mathbf{M_{i+1}} = (\mathbf{M_i} * \mathbf{A_m} + \mathbf{C_m})$ *mod* $\mathbf{R_m}$

Marie's employee ID is 0, and all other employees have IDs from 1 to **N** - 1, inclusive. The salary of employee **i** is $\mathbf{S_i}$. For every employee **i** other than Marie, the manager is $\mathbf{M_i}$ *mod* **i**. (Note that this means that $\mathbf{M_0}$ does not affect Marie's manager -- she has none!)

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 largest number of employees Marie can keep at the company, including herself, such that all of laws 1-7 are obeyed.

Limits

$1 \le \mathbf{T} \le 100$.
$0 \le \mathbf{S_0} < \mathbf{R_s}$.
$0 \le \mathbf{M_0} < \mathbf{R_m}$.
$0 \le \mathbf{A_s}, \mathbf{A_m} \le 1000$.
$0 \le \mathbf{C_s}, \mathbf{C_m} \le 10^9$.

Small dataset

$1 \le \mathbf{N} \le 1000$.
$1 \le \mathbf{D} \le 1000$.
$1 \le \mathbf{R_s}, \mathbf{R_m} \le 1000$.

Large dataset

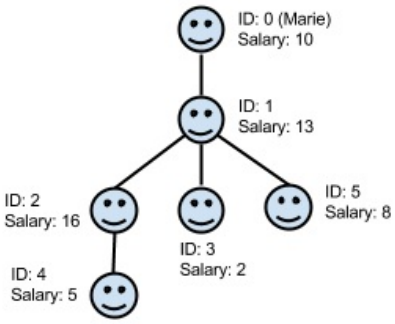
$1 \le \mathbf{N} \le 10^6$.
$1 \le \mathbf{D} \le 10^6$.
$1 \le \mathbf{R_S}, \mathbf{R_M} \le 10^6$.

Sample

Input	Output
3 1 395 18 246 615815 60 73 228 14618 195 6 5 10 1 3 17 5 2 7 19 10 13 28 931 601463 36 231 539 556432 258	Case #1: 1 Case #2: 3 Case #3: 5

In Case #1, the company has only a CEO and no other employees, but it does not violate any of the laws, so no changes need to be made.

Here is the org chart for Case #2:



The optimal strategy is to save employees 0, 1, and 5 (who have salaries of 10, 13, and 8, respectively). It is not possible to save employee 2, for example, because her salary is more than 5 away from employee 0's salary of 10; since employee 0 cannot be laid off, employee 2 must be laid off (along with all employees who report to her).

If you want to check your sequences for employees 1 through 5, they are:

S: 13, 16, 2, 5, 8
M: 17, 3, 13, 14, 16
Manager numbers: 17 % 1 = 0, 3 % 2 = 1, 13 % 3 = 1, 14 % 4 = 2, 16 % 5 = 1

Problem B. Smoothing Window

Small input 6 points	Solve B-small
Large input 7 points	Solve B-large

Problem

Adamma is a climate scientist interested in temperature. Every minute, she records the current temperature as an integer, creating a long list of integers: x_1, x_2, ..., $x_{\mathbf{N}}$. (Adamma uses her own special temperature scale rather than a familiar one like Celsius or Kelvin, so it's possible for the values to be large and negative!) She often plots these temperatures on her computer screen.

This morning, she decided to compute a sliding average of this list in order to get a smoother plot. She used a smoothing window of size **K**, which means that she converted the sequence of **N** temperatures into a sequence of (**N** - **K** + 1) average temperatures: s_1, s_2, ..., $s_{\mathbf{N}-\mathbf{K}+1}$. Each s_i is the average of the values x_i, x_{i+1}, ..., $x_{i+\mathbf{K}-1}$. The original x_i values were all integers, but some of the s_i may be fractional.

Unfortunately, Adamma forgot to save the original sequence of temperatures! And now she wants to answer a different question -- what was the difference between the largest temperature and the smallest temperature? In other words, she needs to compute $\max\{x_1, ..., x_{\mathbf{N}}\} - \min\{x_1, ..., x_{\mathbf{N}}\}$. But she only has **N**, **K**, and the smoothed sequence.

After some thinking, Adamma has realized that this might be impossible because there may be several valid answers. In that case, she wants to know the smallest possible answer among all of the possible original sequences that could have produced her smoothed sequence with the given values of **N** and **K**.

Input

The first line of the input gives the number of test cases, **T**. **T** test cases follow; each test case consists of two lines. The first line contains integers **N** and **K** separated by a space character. The second line contains integer values sum_1, sum_2, ..., $sum_{\mathbf{N}-\mathbf{K}+1}$, separated by space characters. s_i is given by sum_i / **K**.

Output

For each test case, output one line containing "Case #x: y", where x is the test case number (starting from 1) and y is the smallest possible difference between the largest and smallest temperature.

Limits

$1 \le \mathbf{T} \le 100$.
$2 \le \mathbf{K} \le \mathbf{N}$.
The sum_i will be integers between -10000 and 10000, inclusive.

Small dataset

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

Large dataset

$2 \le \mathbf{N} \le 1000$.
$2 \le \mathbf{K} \le 100$.

Sample

Input	Output
3 10 2 1 2 3 4 5 6 7 8 9 100 100 -100 7 3 0 12 0 12 0	Case #1: 5 Case #2: 0 Case #3: 12

In Case #1, the smoothed sequence is:

0.5, 1.0, 1.5, 2.0, 2.5, 3.0, 3.5, 4.0, 4.5

The integer sequence that gives the smallest difference is:

0, 1, 1, 2, 2, 3, 3, 4, 4, 5

Note that the sequence:

0.5, 0.5, 1.5, 1.5, 2.5, 2.5, 3.5, 3.5, 4.5, 4.5

Would give the same smoothed sequence with a maximum difference of 4, but this is not a valid answer because the original temperatures are known to have been integers.

In Case #2, all we know is that the sum of the 100 original values was -100. It's possible that all of the original values were exactly -1, in which case the difference between the largest and smallest temperatures would be 0, which is as small as differences get!

In Case #3, the original sequence could have been:

-4, 8, -4, 8, -4, 8, -4

Problem C. Runaway Quail

Small input 8 points	Solve C-small
Large input 15 points	Solve C-large

Problem

Oh no -- your **N** pet quail have all gotten loose! You are currently at position 0 on a line; the **i**th quail starts off at some nonzero integer (positive or negative) position $\mathbf{P_i}$ on that line, in meters, and will continuously run away from you at a constant integer speed of $\mathbf{S_i}$ meters per second. You can run at a constant integer speed of **Y** meters per second, and can change direction instantaneously whenever you want. Note that quail constantly run away from you even if you are not running toward them at the time. Whenever you occupy the same point as a quail, that quail is caught (this takes no additional time).

What is the minimum number of seconds it will take you to catch all of the quail?

Input

The first line of the input gives the number of test cases, **T**. **T** test cases follow. Each begins with one line with two space-separated integers **Y**, your speed, and **N**, the number of quail, and is followed by two more lines with **N** space-separated integers each. The first of these gives the positions $\mathbf{P_i}$ of the quail, and the second gives the speeds $\mathbf{S_i}$.

Output

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

y will be considered correct if it is within an absolute or relative error of 10^{-6} of the correct answer. See the FAQ for an explanation of what that means, and what formats of real numbers we accept.

Limits

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

$2 \le \mathbf{Y} \le 1000$.
$-10^7 \le \mathbf{P_i} \le 10^7$; no $\mathbf{P_i}$ is 0.
$1 \le \mathbf{S_i} < \mathbf{Y}$.

Small dataset

$1 \le \mathbf{N} \le 25$.

Large dataset

$1 \le \mathbf{N} \le 500$.

Sample

Input	Output
2 4 3 -3 -6 -9 3 2 1 2 2 1 -1 1 1	Case #1: 3.000000 Case #2: 5.000000

In Case #1, you can run to the left and catch all three quail at the same time, 12 meters to the left of the starting position, which takes 3 seconds.

In Case #2, one optimal strategy is to run to the left until the second quail is caught at -2 m, which takes one second, and then run to the right in pursuit of

the first quail, which you will catch at 6 m, taking four more seconds.

Round 3 2015

A. Fairland
B. Smoothing Window
C. Runaway Quail
D. Log Set
E. River Flow

Contest Analysis
Questions asked

Problem D. Log Set

Small input 6 points	Solve D-small
Large input 19 points	Solve D-large

Problem

The *power set* of a set S is the set of all subsets of S (including the empty set and S itself). It's easy to go from a set to a power set, but in this problem, we'll go in the other direction!

We've started with a set of (not necessarily unique) integers S, found its power set, and then replaced every element in the power set with the sum of elements of that element, forming a new set S'. For example, if S = {-1, 1}, then the power set of S is {{}, {-1}, {1}, {-1, 1}}, and so S' = {0, -1, 1, 0}. S' is allowed to contain duplicates, so if S has N elements, then S' always has exactly 2^N elements.

Given a description of the elements in S' and their frequencies, can you determine our original S? It is guaranteed that S exists. If there are multiple possible sets S that could have produced S', we guarantee that our original set S was the *earliest* one of those possibilities. To determine whether a set S_1 is earlier than a different set S_2 of the same length, sort each set into nondecreasing order and then examine the leftmost position at which the sets differ. S_1 is earlier iff the element at that position in S_1 is smaller than the element at that position in S_2.

Input

The first line of the input gives the number of test cases, **T**. **T** test cases follow. Each consists of one line with an integer **P**, followed by two more lines, each of which has **P** space-separated integers. The first of those lines will have all of the different elements E_1, E_2, ..., $E_{\mathbf{P}}$ that appear in S', sorted in ascending order. The second of those lines will have the number of times F_1, F_2, ..., $F_{\mathbf{P}}$ that each of those values appears in S'. That is, for any i, the element E_i appears F_i times in S'.

Output

For each test case, output one line containing "Case #x: ", where x is the test case number (starting from 1), followed by the elements of our original set S, separated by spaces, in nondecreasing order. (You will be listing the elements of S directly, and not providing two lists of elements and frequencies as we do for S'.)

Limits

$1 \le \mathbf{T} \le 100$.
$1 \le \mathbf{P} \le 10000$.
$\mathbf{F_i} \ge 1$.

Small dataset

S will contain between 1 and 20 elements.
0 ≤ each $E_i \le 10^8$.

Large dataset

S will contain between 1 and 60 elements.
$-10^{10} \le$ each $E_i \le 10^{10}$.

Sample

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

```
0 1 3 4
4 4 4 4
3
-1 0 1
1 2 1
5
-2 -1 0 1 2
1 2 2 2 1
```

Note that Cases #4 and #5 are not within the limits for the Small dataset.

In Case #4, S = {-1, 1} is the only possible set that satisfies the conditions. (Its subsets are {}, {-1}, {1}, and {-1, 1}. Those have sums 0, -1, 1, and 0, respectively, so S' has one copy of -1, two copies of 0, and one copy of 1, which matches the specifications in the input.)

For Case #5, note that S = {-1, -1, 2} also produces the same S' = {-2, -1, -1, 0, 0, 1, 1, 2}, but S = {-2, 1, 1} is earlier than {-1, -1, 2}, since at the first point of difference, -2 < -1. So `-1 -1 2` would **not** be an acceptable answer. `1 -2 1` would also be unacceptable, even though it is the correct set, because the elements are not listed in nondecreasing order.

Problem E. River Flow

Small input 10 points	Solve E-small
Large input 17 points	Solve E-large

Problem

The city you live in lies on the banks of the spectacular Binary river. The water in the river comes from some tributary streams that start way up in the mountains. Unfortunately for your city, there are farmers who live in the mountains who need to use up some of the water in the tributary streams for their crops.

Long ago, the city struck a deal with the farmers to allow them to farm while keeping the river flowing: each farmer was allowed to use the water for her crops exactly half the time. The farmers would alternately divert water for their crops for a day and leave the water to run down the river for a day. The result was a disaster! Because the farmers' water usage was synchronized, with everyone either diverting or not diverting water at the same time, the river would run dry every other day and then flood the city the next.

To solve this problem, the city went back to the farmers and asked each one to choose some integer power of 2 (this is the Binary River after all) between 1 and **D**, inclusive, and toggle her water usage (either start or stop collecting water) every time that number of days has elapsed. (Not every power of 2 between 1 and **D** was necessarily represented, and multiple farmers may have selected the same integer. 1 counts as a power of 2.) The idea was that this would make the water usage more even overall, and so the droughts and flooding would become less frequent.

This all happened a long time ago, and you and the other citizens have recently become suspicious that the farmers aren't sticking to the agreement. (You're not even sure how many farmers there are right now!) However, the only data you have is **N** days' history of the amount of water flowing through the city. Can you tell if the farmers are being honest?

Each tributary stream has flow 1 and the flow through the main river is the sum of all the tributary streams that are not being diverted for farming. (Before looking at the records, you don't know how many tributary streams there are.) At most 1 farmer will divert the water from each tributary stream, but there may be some tributary streams from which no farmers ever divert water. Note that the farmers started their water diversion cycles long before the city started recording the water flow, but there is no guarantee that they all started on the same day.

Input

The first line of the input gives the number of test cases, **T**. **T** test cases follow. Each test case starts with a line containing two space-separated integers **N** and **D**. The next line contains **N** space-separated integers, with the ith integer $\mathbf{d_i}$ giving the river flow on the ith day.

Output

For each case, output one line containing "Case #x: **M**", where x is the test case number (starting from 1) and **M** is the smallest number of farmers who could be diverting water from the streams according to the described model, consistent with the observed flow through the river.

If you are sure that at least one farmer is active, but there is no way that the supplied input could be explained by farmers obeying the rules, then output CHEATERS! instead of a number.

Limits

$1 \le \mathbf{T} \le 50$.
D will be a power of 2.
$1 \le \mathbf{D} \le \text{floor}(\mathbf{N} / 2)$.

Small dataset

$1 \le \mathbf{N} \le 50$.
$0 \le \mathbf{d_i} \le 5$.

Large dataset

$1 \le \mathbf{N} \le 5000$.

$0 \leq \mathbf{d_i} \leq 1000$.

Sample

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

Explanation

Case #1 is consistent with two tributary streams with no farmers drawing from them.

Case #2 could a single tributary stream being diverted every 4 days. However, **D** is 2 in this case, so this farmer is breaking the agreement.

Case #3 could be two farmers each with a diversion cycle of 4 days.

Case #4 could be three farmers with diversion cycles of 1, 2 and 4 days.

Problem A. Costly Binary Search

Small input 8 points	Solve A-small
Large input 19 points	Solve A-large

Problem

You were asked to implement arguably the most important algorithm of all: binary search. More precisely, you have a sorted array of objects, and a new object that you want to insert into the array. In order to find the insertion position, you can compare your object with the objects in the array. Each comparison can return either "greater", meaning that your object should be inserted to the right of the compared object, or "less", meaning that your object should be inserted to the left of the compared object. For simplicity, comparisons never return "equal" in this problem. It is guaranteed that when your object is greater than some object in the array, it is also greater than all objects to the left of that object; similarly, when your object is less than some object of the array, it is also less than all objects to the right of that object. If the array has **n** elements, there are **n**+1 possible outcomes for your algorithm.

In this problem, not all comparisons have the same cost. More precisely, comparing your object with **i**-th object in the array costs $\mathbf{a_i}$, an integer between 1 and 9, inclusive.

What will be the total cost, in the worst case, of your binary search? Assume you follow an optimal strategy and try to minimize the total cost in the worst case.

Input

The first line of the input gives the number of test cases, **T**. **T** lines follow. Each of those lines contains one sequence of digits describing the comparison costs $\mathbf{a_i}$ for one testcase. The size of the array **n** is given by the length of this sequence.

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 total binary search cost in the worst case.

Limits

$1 \le \mathbf{T} \le 50$.
All digits are between 1 and 9, inclusive.
There are no spaces between digits on one line.

Small dataset

$1 \le \mathbf{n} \le 10^4$.

Large dataset

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

Sample

Input	Output
4 111 1111 1111111 1111119	Case #1: 2 Case #2: 3 Case #3: 3 Case #4: 10

Problem B. Campinatorics

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 6 points	Solve B-small
Large input 21 points	Solve B-large

Problem

"Summer is finally here: time to relax, have some fun, go outside and enjoy the nice weather!" says Alice, a very dedicated Ranger working in a popular National Park. During the summer, lots of families take some time off to camp there and have a good time, and it is Alice's job to accommodate the visitors.

Alice is in charge of one of the many camps around the park. The camp can be described as a matrix of size **N** x **N**, where each cell has enough space for at most one tent. In order to arrange the families in the camp, there are several regulations that Alice needs to follow:

- Only families with **1, 2 or 3** members are allowed in the camp. Also, each tent can contain members of only one family, and families cannot be split across multiple tents.
- For security reasons, Alice doesn't want the rows or columns to be too crowded or too empty, so she wants exactly **3** members in each row and column.
- Also, according to the park's safety policies, there shouldn't be more than **2** tents in any row or column.

Additionally, Alice knows in advance that at least **X** three-member families will be visiting the camp, and that there will be enough one- or two-member families to fill the rest of the camp.

For example, these are valid arrangements for N = 3 and X = 0:

```
1 2 0 | 3 0 0
0 1 2 | 0 1 2
2 0 1 | 0 2 1
```

These are not valid arrangements for N = 3 and X = 1:

```
1 2 0 | 0 3 0 | 1 2 0 | 1 1 1
0 1 2 | 3 0 0 | 0 2 0 | 1 1 1
2 0 1 | 0 0 0 | 2 0 1 | 1 1 1
```

- The first one is not valid because there should be at least one three-member family.
- The second example is not valid because the number of persons in the third row (and column) is not three.
- The third one is invalid because there are more than three members in the second column (and fewer than three in the second row).
- The last example contains more than two tents per row or column.

Finally, Alice likes to keep things interesting. She would like to know how many different arrangements are possible given **N** and **X**.

Two arrangements A and B are considered different, if a cell in one arrangement contains a tent, but the same cell in the other arrangement doesn't; or if there is a tent in the same cell of both arrangements, but the number of members in that cell in A is different than the number of members in the same cell in B.

Input

The first line of the input contains **T**, the number of test cases. **T** test cases follow. Each test case consists of exactly one line with two integers **N** and **X** corresponding to the number of rows (and columns) in Alice's camp and the minimum number of three-member families, respectively.

Output

For each test case, output one line containing "Case #X: Y", where X is the test case number (starting from 1) and Y is the number of possible arrangements.

The answer may be huge, so output the answer **modulo $10^9 + 7$**.

Limits

$1 \le \mathbf{T} \le 200$.
$0 \le \mathbf{X} \le N$.

Small dataset

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

Large dataset

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

Sample

```
Input     Output

3         Case #1: 2
2 2       Case #2: 24
3 1       Case #3: 738721209
15 0
```

In case #1, you have two different valid arrangements:

```
0 3  |  3 0
3 0  |  0 3
```

Problem C. Pretty Good Proportion

Small input 5 points	Solve C-small
Large input 22 points	Solve C-large

Problem

I have a sequence of **N** binary digits. I am looking for a substring with just the right proportion of 0s and 1s, but it may not exist, so I will settle for something that's just pretty good.

Can you find a substring where the fraction of 1s is as close as possible to the given fraction **F**? Output the earliest possible index at which such a substring starts.

Input

The first line of the input gives the number of test cases, **T**. **T** test cases follow. Each one starts with a line containing **N** and **F**. **F** will be a decimal fraction between 0 and 1 inclusive, with exactly 6 digits after the decimal point. The next line contains **N** digits, each being either 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 0-based index of the start of the substring with the fraction of 1s that is as close as possible to **F**. If there are multiple possible answers, output the smallest correct value.

Limits

$1 \le \mathbf{T} \le 100$.
$0 \le \mathbf{F} \le 1$
F will have exactly 6 digits after the decimal point.

Small dataset

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

Large dataset

$1 \le \mathbf{N} \le 500{,}000$.

Sample

Input	Output
5 12 0.666667 001001010111 11 0.400000 10000100011 9 0.000000 111110111 5 1.000000 00000 15 0.333333 000000000011000	Case #1: 5 Case #2: 5 Case #3: 5 Case #4: 0 Case #5: 6

In Case #1, there is no substring that has exactly a 1-proportion of exactly 666667/1000000. The closest we can get is 2/3. The input string has 5 substrings that achieve it -- 3 substrings of length 3 that start at indices 5, 7, and 8 (101, 101, and 011); as well as two substrings of length 6 that start at indices 5 and 6 (101011 and 010111). The smallest of these indices is 5.

Problem D. Taking Over The World

Small input 7 points	Solve D-small
Large input 29 points	Solve D-large

Problem

You and your friend Pinky have a plan to take over the world. But first you need to disable a certain secret weapon.

It is hidden inside a twisted maze of passages (a graph) with one entrance. Pinky is going to be at the vertex with the secret weapon, disabling it. Meanwhile, a security team at the graph entrance will be alerted, and will run through the graph to try to get to Pinky in time to stop him. You are going to be slowing down the security team to give Pinky as much time as possible. It takes one unit of time to traverse any edge of the graph, but you can additionally "obstruct" up to **K** vertices. It takes one additional unit of time to traverse an obstructed vertex. You will choose to obstruct a set of vertices that slows down the security team by as much as possible.

If the security team will be starting at the graph entrance and is trying to get to the secret weapon vertex, how much time will it take them to get there? Note that you have to commit all your obstructions before the security guards start their journey, and the security guards will know which vertices you have obstructed and will choose an optimal path based on that information.

Obstructing the secret weapon vertex is not useful because that will not delay the guards any further after they have already caught Pinky. Obstructing the entrance, on the other hand, is obviously a good idea.

Input

The first line of the input gives the number of test cases, **T**. **T** test cases follow. Each one starts with a line containing **N**, **M**, and **K**. The next **M** lines each contain a pair of vertices connected by an edge. Vertices are numbered from 0 (the entrance) to **N** - 1 (the secret weapon room). The first vertex will always be smaller than the second vertex, and no pair of vertices will appear more than once in the same test case. Edges are bi-directional -- the guards can travel along any edge in either direction.

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 time it will take the security guards to get from the entrance to the secret weapon room.

Limits

$1 \le \mathbf{T} \le 100$.
$2 \le \mathbf{N} \le 100$.
$1 \le \mathbf{M} \le \mathbf{N} * (\mathbf{N} - 1) / 2$.
$1 \le \mathbf{K} \le \mathbf{N}$.
There will always be a path from room 0 to room **N** - 1.

Small dataset

It will not be possible to delay the guards by more than 2 time units, compared to the shortest unobstructed path length (using the given **K**).

Large dataset

No extra restrictions.

Sample

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

```
4 4 2
0 1
0 2
1 3
2 3
7 11 3
0 1
0 2
0 3
1 4
1 5
2 4
2 5
3 4
3 5
4 6
5 6
```

Problem E. Merlin QA

Small input 8 points	Solve E-small
Large input 30 points	Solve E-large

Problem

Edythe is a young sorceress working in the quality assurance department of Merlin, Inc. -- a magic spell factory. Her job is to test the magic spells that Merlin himself invents. Each spell requires precise amounts of certain ingredients and transforms them into other amounts of other ingredients. Edythe's job is to cast each spell exactly once in order to verify that the spell works correctly.

She can cast a spell only if she has the required amount of each ingredient. If she has already created ingredients of the right type from previous spells, Edythe must use those first. However, if she still needs more ingredients, she is allowed to take them from Merlin's storehouse. She has no ingredients to start with, but at the end, she gets to keep all the extra ingredients that she created and didn't use.

Edythe wants to make as much profit as possible from her apprenticeship! She has to cast each of the **N** given spells exactly once, but she is free to do so in any order. Assuming that each spell works as expected, which ordering lets her earn the most money in the end?

For example, imagine that the test plan contains the following 3 spells:

1. Inputs: $7 worth of gold. Outputs: $5 worth of sulfur.
2. Inputs: nothing. Outputs: $10 worth of gold, $10 worth of sulfur.
3. Inputs: $3 worth of gold, $20 worth of sulfur. Outputs: $2 worth of toads.

Note that the first spell converts gold into sulfur, the second spell conjures up gold and sulfur from nothing, and the third spell converts gold and sulfur into toads.

If Edythe were to cast these spells in the order 1, 2, 3, then she would start by fetching $7 worth of gold from the storehouse for spell #1. That would let her cast spells #1 and #2, giving her $10 worth of gold and $15 worth of sulfur. For the final spell, she would need $3 worth of gold and $20 worth of sulfur. She would have to use all of the sulfur she created so far, $3 worth of gold, and $5 more worth of sulfur that she fetched from the storehouse. This would leave her with $9 worth of ingredients at the end ($7 worth of gold and $2 worth of toads).

But there is a better plan. If she cast the spells in the order 3, 1, 2, she would have $27 worth of ingredients at the end ($10 worth of gold, $15 worth of sulfur, and $2 worth of toads).

Input

The first line of the input gives the number of test cases, **T**. **T** test cases follow. Each one starts with a line containing **N** and **M**. **M** is the number of kinds of ingredients in the world. Each of the next **N** lines contains **M** integers describing a spell. Each integer is the value (or cost) of the corresponding ingredient. Negative integers are the dollar costs of the input ingredients; positive integers are the dollar values of the output ingredients; and zeros denote ingredients that are neither produced nor consumed by the spell. This also implies that no spell can simultaneously consume and produce the same ingredient.

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 largest value of ingredients Edythe can have at the end.

Limits

$1 \leq \mathbf{T} \leq 100$.
$1 \leq \mathbf{N} \leq 100$.
$-100 \leq$ Each integer in each spell ≤ 100.

Small dataset

$1 \leq \mathbf{M} \leq 2$.

Large dataset

$1 \le \mathbf{M} \le 8$.

Sample

Input	Output
2 3 1 1 0 -1 3 3 -7 5 0 10 10 0 3 -20 2	Case #1: 1 Case #2: 27

Problem F. Crane Truck

Small input 8 points	Solve F-small
Large input 37 points	Solve F-large

Problem

You are in a large storage facility, with 2^{40} storage locations arranged in a circle.

A truck with a crane on it moves along the circle of storage locations, picking up or putting down crates according to a program. (The truck has an unlimited supply of crates on board, so it can always put more crates down.)

The program consists of a sequence of these instructions:

- b : move back one location
- f : move forward one location
- u : pick up one crate at the current location
- d : put down one crate at the current location
- (: do nothing
-) : if there is more than one crate at the current location, move back to the most recent (in the sequence of instructions, and continue the program from there. (This doesn't move the truck.)

(and) instructions in the program will always come in pairs: a (will be followed later by a matching). There will be at most two such pairs in the program, and if there are two pairs, they will not be nested – that is, there will be either:

- no (or) instructions;
- one (instruction somewhere in the program, followed later by one) instruction;
- a (instruction, followed later by a) instruction, followed later by another (, and again later by another).

The sample cases contain examples of each of these.

Each storage location begins with one crate, before the crane truck starts running its program.

Mysteriously, if the truck picks up the last crate at a location, another truck instantly comes along and puts down 256 crates there! Similarly, if the truck puts down a crate at a location, and that location then has 257 crates, another truck instantly drives past and picks up 256 of the crates, leaving one behind! So every location always has between 1 and 256 crates.

How many times will the truck move forward or backward before reaching the end of its program?

Input

One line containing an integer **T**, the number of test cases in the program.

T lines, each containing a crane truck program with up to 2000 characters.

Output

T lines, one for each test case, containing "Case #**X**: **Y**" where **X** is the test case number, and **Y** is the number of times the truck moves.

Limits

$1 \le \mathbf{T} \le 20$.
$1 \le$ the length of the program ≤ 2000.
The program is guaranteed to terminate.

Small dataset

The program will contain at most one pair of (and) instructions.

Large dataset

The program will contain at most two pairs of (and) instructions.

Sample

Input	Output
4 ufffdddbbbdd dddd(fdbu)fff dddd(fddddbu)f(fdddddbu) bf	Case #1: 6 Case #2: 11 Case #3: 49 Case #4: 2