Problem A. Bot Trust

Small input
10 points
Solve A-small

Large input
10 points
Solve A-large

Problem

Blue and Orange are friendly robots. An evil computer mastermind has locked them up in separate hallways to test them, and then possibly give them cake.

Each hallway contains 100 buttons labeled with the positive integers {1, 2, ..., 100}. Button k is always k meters from the start of the hallway, and the robots both begin at button 1. Over the period of one second, a robot can walk one meter in either direction, or it can press the button at its position once, or it can stay at its position and not press the button. To complete the test, the robots need to push a certain sequence of buttons in a certain order. Both robots know the full sequence in advance. How fast can they complete it?

For example, let's consider the following button sequence:

```
O 2, B 1, B 2, O 4
```

Here, O 2 means button 2 in Orange's hallway, B 1 means button 1 in Blue's hallway, and so on. The robots can push this sequence of buttons in 6 seconds using the strategy shown below:

```
Time | Orange           | Blue
-----+------------------+-----------------
  1  | Move to button 2 | Stay at button 1
  2  | Push button 2    | Stay at button 1
  3  | Move to button 3 | Push button 1
  4  | Move to button 4 | Move to button 2
  5  | Stay at button 4 | Push button 2
  6  | Push button 4    | Stay at button 2
```

Note that Blue has to wait until Orange has completely finished pushing O 2 before it can start pushing B 1.

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 beginning with a positive integer **N**, representing the number of buttons that need to be pressed. This is followed by **N** terms of the form "$\mathbf{R}_i$ $\mathbf{P}_i$" where $\mathbf{R}_i$ is a robot color (always 'O' or 'B'), and $\mathbf{P}_i$ is a button position.

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 seconds required for the robots to push the given buttons, in order.

Limits

$1 \leq \mathbf{P}_i \leq 100$ for all i.

Small dataset

$1 \leq \mathbf{T} \leq 20$.
$1 \leq \mathbf{N} \leq 10$.

Large dataset

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

Sample

Input	Output
3	Case #1: 6
4 O 2 B 1 B 2 O 4	Case #2: 100
3 O 5 O 8 B 100	Case #3: 4
2 B 2 B 1	

Problem B. Magicka

Small input
10 points
Solve B-small

Large input
15 points
Solve B-large

Introduction

Magicka™ is an action-adventure game developed by Arrowhead Game Studios. In Magicka you play a wizard, invoking and combining elements to create Magicks. This problem has a similar idea, but it does not assume that you have played Magicka.

Note: "invoke" means "call on." For this problem, it is a technical term and you don't need to know its normal English meaning.

Problem

As a wizard, you can **invoke** eight elements, which are the "base" elements. Each base element is a single character from {Q, W, E, R, A, S, D, F}. When you invoke an element, it gets appended to your **element list**. For example: if you invoke W and then invoke A, (we'll call that "invoking WA" for short) then your element list will be [W, A].

We will specify pairs of base elements that **combine** to form non-base elements (the other 18 capital letters). For example, Q and F might combine to form T. If the two elements from a pair appear at the end of the element list, then both elements of the pair will be immediately removed, and they will be replaced by the element they form. In the example above, if the element list looks like [A, Q, F] or [A, F, Q] at any point, it will become [A, T].

We will specify pairs of base elements that are **opposed** to each other. After you invoke an element, if it isn't immediately combined to form another element, and it is opposed to something in your element list, then your whole element list will be cleared.

For example, suppose Q and F combine to make T. R and F are opposed to each other. Then invoking the following things (in order, from left to right) will have the following results:

- QF → [T] (Q and F combine to form T)
- QEF → [Q, E, F] (Q and F can't combine because they were never at the end of the element list together)
- RFE → [E] (F and R are opposed, so the list is cleared; then E is invoked)
- REF → [] (F and R are opposed, so the list is cleared)
- RQF → [R, T] (QF combine to make T, so the list is not cleared)
- RFQ → [Q] (F and R are opposed, so the list is cleared)

Given a list of elements to invoke, what will be in the element list when you're done?

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 the following space-separated elements in order:

First an integer **C**, followed by **C** strings, each containing three characters: two base elements followed by a non-base element. This indicates that the two base elements combine to form the non-base element. Next will come an integer **D**, followed by **D** strings, each containing two characters: two base elements that are opposed to each other. Finally there will be an integer **N**, followed by a single string containing **N** characters: the series of base elements you are to invoke. You will invoke them in the order they appear in the string (leftmost character first, and so on), one at a time.

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 list in the format "[e_0, e_1, ...]" where e_i is the i^{th} element of the final element list. Please see the sample output for examples.

Limits

$1 \le \mathbf{T} \le 100$.
Each pair of base elements may only appear together in one combination, though they may appear in a combination and also be opposed to each other.
No base element may be opposed to itself.

Unlike in the computer game Magicka, there is no limit to the length of the element list.

Small dataset

$0 \leq \mathbf{C} \leq 1$.
$0 \leq \mathbf{D} \leq 1$.
$1 \leq \mathbf{N} \leq 10$.

Large dataset

$0 \leq \mathbf{C} \leq 36$.
$0 \leq \mathbf{D} \leq 28$.
$1 \leq \mathbf{N} \leq 100$.

Sample

Input	Output
5 0 0 2 EA 1 QRI 0 4 RRQR 1 QFT 1 QF 7 FAQFDFQ 1 EEZ 1 QE 7 QEEEERA 0 1 QW 2 QW	Case #1: [E, A] Case #2: [R, I, R] Case #3: [F, D, T] Case #4: [Z, E, R, A] Case #5: []

Problem C. Candy Splitting

Small input
10 points
Solve C-small

Large input
15 points
Solve C-large

Problem

Sean and Patrick are brothers who just got a nice bag of candy from their parents. Each piece of candy has some positive integer value, and the children want to divide the candy between them. First, Sean will split the candy into two piles, and choose one to give to Patrick. Then Patrick will try to calculate the value of each pile, where the value of a pile is the sum of the values of all pieces of candy in that pile; if he decides the piles don't have equal value, he will start crying.

Unfortunately, Patrick is very young and doesn't know how to add properly. He *almost* knows how to add numbers in binary; but when he adds two 1s together, he always forgets to carry the remainder to the next bit. For example, if he wants to sum 12 (1100 in binary) and 5 (101 in binary), he will add the two rightmost bits correctly, but in the third bit he will forget to carry the remainder to the next bit:

```
  1100
+ 0101
------
  1001
```

So after adding the last bit without the carry from the third bit, the final result is 9 (1001 in binary). Here are some other examples of Patrick's math skills:

```
5 + 4 = 1
7 + 9 = 14
50 + 10 = 56
```

Sean is very good at adding, and he wants to take as much value as he can without causing his little brother to cry. If it's possible, he will split the bag of candy into two non-empty piles such that Patrick thinks that both have the same value. Given the values of all pieces of candy in the bag, we would like to know if this is possible; and, if it's possible, determine the maximum possible value of Sean's pile.

Input

The first line of the input gives the number of test cases, **T**. **T** test cases follow. Each test case is described in two lines. The first line contains a single integer **N**, denoting the number of candies in the bag. The next line contains the **N** integers $\mathbf{C_i}$ separated by single spaces, which denote the value of each piece of candy in the bag.

Output

For each test case, output one line containing "Case #x: y", where x is the case number (starting from 1). If it is impossible for Sean to keep Patrick from crying, y should be the word "NO". Otherwise, y should be the value of the pile of candies that Sean will keep.

Limits

$1 \le \mathbf{T} \le 100$.
$1 \le \mathbf{C_i} \le 10^6$.

Small dataset

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

Large dataset

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

Sample

Input	Output

```
2           Case #1: NO
5           Case #2: 11
1 2 3 4 5
3
3 5 6
```

Problem D. GoroSort

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

Problem

Goro has 4 arms. Goro is very strong. You don't mess with Goro. Goro needs to sort an array of **N** different integers. Algorithms are not Goro's strength; strength is Goro's strength. Goro's plan is to use the fingers on two of his hands to hold down several elements of the array and hit the table with his third and fourth fists as hard as possible. This will make the unsecured elements of the array fly up into the air, get shuffled randomly, and fall back down into the empty array locations.

Goro wants to sort the array as quickly as possible. How many hits will it take Goro to sort the given array, on average, if he acts intelligently when choosing which elements of the array to hold down before each hit of the table? Goro has an infinite number of fingers on the two hands he uses to hold down the array.

More precisely, before each hit, Goro may choose any subset of the elements of the array to freeze in place. He may choose differently depending on the outcomes of previous hits. Each hit permutes the unfrozen elements uniformly at random. Each permutation is equally likely.

Input

The first line of the input gives the number of test cases, **T**. **T** test cases follow. Each one will consist of two lines. The first line will give the number **N**. The second line will list the **N** elements of the array in their initial order.

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 expected number of hit-the-table operations when following the best hold-down strategy. Answers with an absolute or relative error of at most 10^{-6} will be considered correct.

Limits

$1 \le \mathbf{T} \le 100$;
The second line of each test case will contain a permutation of the **N** smallest positive integers.

Small dataset

$1 \le \mathbf{N} \le 10$;

Large dataset

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

Sample

Input	Output
3 2 2 1 3 1 3 2 4 2 1 4 3	Case #1: 2.000000 Case #2: 2.000000 Case #3: 4.000000

Explanation

In test case #3, one possible strategy is to hold down the two leftmost elements first. Elements 3 and 4 will be free to move. After a table hit, they will land in the correct order [3, 4] with probability 1/2 and in the wrong order [4, 3] with probability 1/2. Therefore, on average it will take 2 hits to arrange them in the correct order. After that, Goro can hold down elements 3 and 4 and hit the table until 1 and 2 land in the correct order, which will take another 2 hits, on average. The total is then 2 + 2 = 4 hits.

Problem A. FreeCell Statistics

Small input 6 points	Solve A-small
Large input 14 points	Solve A-large

Problem

I played **D** (**D** > 0) games of FreeCell today. Each game of FreeCell ends in one of two ways -- I either win, or I lose. I've been playing for many years, and have so far played **G** games in total (obviously, **G** ≥ **D**).

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

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

Input

The first line of the input gives the number of test cases, **T**. **T** lines follow. Each line contains 3 integers -- **N**, $\mathbf{P_D}$ and $\mathbf{P_G}$.

Output

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

Limits

0 ≤ $\mathbf{P_D}$ ≤ 100;
0 ≤ $\mathbf{P_G}$ ≤ 100.

Small dataset

1 ≤ **T** ≤ 100;
1 ≤ **N** ≤ 10.

Large dataset

1 ≤ **T** ≤ 2000;
1 ≤ **N** ≤ 10^{15}.

Sample

Input	Output
3 1 100 50 10 10 100 9 80 56	Case #1: Possible Case #2: Broken Case #3: Possible

In Case #3, I could have played 5 games today (**D** = 5) and 25 games in total (**G** = 25), and won 4 games today (80% of 5) and 14 games in total (56% of 25).


Powered by
Google Cloud Platform

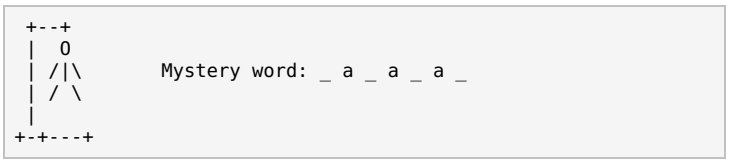
Problem B. The Killer Word

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 B-small
Large input 20 points	Solve B-large

Problem

You are playing Hangman with your friend Sean. And while you have heard that Sean is very good at taking candy from a baby, he is not as good at this game. Can you take advantage of Sean's imperfect strategy, and make him lose as badly as possible?



Hangman is played as follows:

- There is a dictionary **D** of all valid words, which both you and Sean know. A word consists only of the characters a - z. In particular, there are no spaces.
- You begin by choosing any word from **D**, and writing it down on a blackboard with each letter replaced by a blank: _.
- On his turn, Sean can choose any letter and ask you if it is in the word. If it is, you reveal *all* locations of that letter. Otherwise, Sean loses a point.
- Once all letters in the word have been revealed, the round ends.
- The round never ends early, no matter how many points Sean loses.

Sean uses a very simple strategy. He makes a list **L** of the 26 letters in some order, and goes through the list one letter at a time. If there is at least one word in **D** that (a) has the letter he is thinking of, and (b) is consistent with what you have written down so far *and the result of all of Sean's previous guesses*, then Sean guesses that letter. Otherwise, he skips it. No matter what, Sean then moves on to the next letter in his list.

Given Sean's list, what word should you choose to make Sean lose as many as points as possible? If several choices are equally good, you should choose the one that appears first in **D**.

Example

Suppose Sean decides to guess the letters in alphabetical order (i.e., **L** = "abcdefghijklmnopqrstuvwxyz"), and **D** contains the words banana, caravan, and pajamas. If you choose pajamas, the game would play out as follows:

- You begin by writing 7 blanks _ _ _ _ _ _ _ on the blackboard. Based on the number of blanks, Sean knows immediately that the word is either caravan or pajamas.
- Sean begins by guessing a since it is first in **L**, and you reveal all locations of the letter a on the blackboard: _ a _ a _ a _.
- Sean skips b even though it is used in banana. Sean already knows that is not your word.
- He then guesses c because it appears in caravan. It does not appear in the word you actually chose though, so Sean loses a point and nothing more is revealed.
- By process of elimination, Sean now knows your word has to be pajamas, so he proceeds to guess j, m, p, and s in order, without losing any more points.

So Sean loses one point if you choose pajamas. He would have gotten either of the other words without losing any points.

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 integers **N** and **M**, representing the number of words in the dictionary and the number of lists to consider.

The next **N** lines contain the words in the dictionary, one per line: $\mathbf{D}_1$, $\mathbf{D}_2$, ..., $\mathbf{D}_\mathbf{N}$. Each word is an arbitrary sequence of characters a - z.

The final **M** lines contain all of the lists Sean will use, one per line: $\mathbf{L}_1$, $\mathbf{L}_2$, ...,

$\mathbf{L_M}$. Each list is exactly 26 letters long, containing each letter exactly once. Sean will use these lists to guess letters as described above.

Output

For each test case, output one line containing "Case #x: $\mathbf{w_1}$ $\mathbf{w_2}$ ... $\mathbf{w_M}$", where x is the case number (starting from 1) and $\mathbf{w_i}$ is the word you should choose if Sean guesses the letters in order $\mathbf{L_i}$. If multiple words cause Sean to lose the same number of points, you should choose the option that appears first in the dictionary.

Limits

$1 \leq \mathbf{T} \leq 10$.
Each word in **D** will have between 1 and 10 characters inclusive.
No two words in **D** will be the same within a single test case.

Small dataset

$1 \leq \mathbf{N} \leq 100$.
$1 \leq \mathbf{M} \leq 10$.

Large dataset

$1 \leq \mathbf{N} \leq 10000$.
$1 \leq \mathbf{M} \leq 100$.

Sample

Input	Output
2 3 2 banana caravan pajamas abcdefghijklmnopqrstuvwxyz etaoisnhrdlcumwfgypbvkjxqz 4 1 potato tomato garlic pepper zyxwvutsrqponmlkjihgfedcba	Case #1: pajamas caravan Case #2: garlic

Problem C. Pseudominion

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

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

Problem

You are playing a game with a fancy deck of cards. Each card has three bonus numbers: a *card bonus* **c**, a *score bonus* **s**, and a *turn bonus* **t**. Some of the cards start in your hand, while the rest are in a deck on the table. You start with one turn.

On each turn, you can choose any card from your hand and play it. If it has bonus numbers **c**, **s**, **t**, then the following happens:

- The card is discarded from your hand, and it can never be used again.
- You draw the first **c** cards from the deck into your hand. If the deck has fewer than **c** cards in it, you draw all of them.
- Your total score increases by **s**.
- Your number of remaining turns increases by **t**.

If you do not have any cards in your hand at the start of a turn, then nothing happens on that turn. Your goal is to get as high a score as possible before running out of turns.

For example, suppose your hand and deck contain the following cards:

```
         +---+---+---+            +---+---+---+
   HAND: | c | s | t |      DECK: | c | s | t |
         +---+---+---+            +---+---+---+
Card #1: | 0 | 0 | 2 |   Card #4: | 1 | 1 | 0 |
Card #2: | 0 | 5 | 0 |   Card #5: | 0 | 1 | 1 |
Card #3: | 2 | 1 | 1 |   Card #6: | 2 | 2 | 0 |
         +---+---+---+            +---+---+---+
```

The following table shows how you can get a score of 8 from these cards. The first three columns show your hand, the number of turns left, and your score before playing each card, and the final column shows which card to play.

```
+---------+------------+-------+------+
| Hand    | Turns left | Score | Play |
+---------+------------+-------+------+
| 1, 2, 3 |      1     |   0   |   1  |
| 2, 3    |      2     |   0   |   3  |
| 2, 4, 5 |      2     |   1   |   2  |
| 4, 5    |      1     |   6   |   5  |
| 4       |      1     |   7   |   4  |
| 6       |      0     |   8   |   -  |
+---------+------------+-------+------+
```

As you can see, the card bonuses and turn bonuses allow you to chain together a long sequence of cards before you have to stop.

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 **N**, the number of cards in your hand. The next **N** lines each contain three integers, **c**, **s**, and **t**, representing the bonus numbers for a single card in your hand.

This is followed by a single line containing **M**, the number of cards in the deck. The next **M** lines each contain three integers, **c**, **s**, and **t**, representing the bonus numbers for a single card in the deck. These cards are listed in the same order in which you draw them.

Output

For each test case, output one line containing "Case #x: S", where S is the largest score you can obtain before running out of turns.

Limits

$1 \le \mathbf{T} \le 100$.
$1 \le \mathbf{N}$.
$0 \le \mathbf{M}$.

N + **M** ≤ 80.

Small dataset

0 ≤ **c** ≤ 1.
0 ≤ **s** ≤ 20.
0 ≤ **t** ≤ 20.

Large dataset

0 ≤ **c** ≤ 2.
0 ≤ **s** ≤ 50.
0 ≤ **t** ≤ 50.

Sample (Small dataset)

```
Input     Output

2         Case #1: 6
4         Case #2: 8
1 0 0
1 1 1
0 5 0
1 2 0
0
2
1 1 1
0 6 0
1
0 1 3
```

Sample (Large dataset)

```
Input     Output

1         Case #1: 8
3
0 0 2
0 5 0
2 1 1
3
1 1 0
0 1 1
2 2 0
```

Problem A. RPI

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

Problem

In the United States, 350 schools compete every year for an invitation to the NCAA College Basketball Tournament. With so many schools, how do you decide who should be invited? Most teams never play each other, and some teams have a much more difficult schedule than others.

Here is an example schedule for 4 teams named A, B, C, D:

```
  |ABCD
-+----
A|.11.
B|0.00
C|01.1
D|.10.
```

Each 1 in a team's row represents a win, and each 0 represents a loss. So team C has wins against B and D, and a loss against A. Team A has wins against B and C, but has not played D.

The NCAA tournament committee uses a formula called the RPI (Ratings Percentage Index) to help rank teams. Traditionally, it has been defined as follows:

```
RPI = 0.25 * WP + 0.50 * OWP + 0.25 * OOWP
```

WP, OWP, and OOWP are defined for each team as follows:

- WP (Winning Percentage) is the fraction of your games that you have won.
  In the example schedule, team A has WP = 1, team B has WP = 0, team C has WP = 2/3, and team D has WP = 0.5.
- OWP (Opponents' Winning Percentage) is the average WP of all your opponents, after first throwing out the games they played against you.
  For example, if you throw out games played against team D, then team B has WP = 0 and team C has WP = 0.5. Therefore team D has OWP = 0.5 * (0 + 0.5) = 0.25. Similarly, team A has OWP = 0.5, team B has OWP = 0.5, and team C has OWP = 2/3.
- OOWP (Opponents' Opponents' Winning Percentage) is the average OWP of all your opponents. OWP is exactly the number computed in the previous step.
  For example, team A has OOWP = 0.5 * (0.5 + 2/3) = 7/12.

Putting it all together, we see team A has RPI = (0.25 * 1) + (0.5 * 0.5) + (0.25 * 7 / 12) = 0.6458333...

There are some pretty interesting questions you can ask about the RPI. Is it a reasonable measure of team's ability? Is it more important for teams to win games, or to schedule strong opponents? These are all good questions, but for this problem, your task is more straightforward: given a schedule of games, can you calculate every team's RPI?

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 the number of teams **N**.

The next **N** lines each contain exactly **N** characters (either '0', '1', or '.') representing a schedule in the same format as the example schedule above. A '1' in row i, column j indicates team i beat team j, a '0' in row i, column j indicates team i lost to team j, and a '.' in row i, column j indicates team i never played against team j.

Output

For each test case, output **N** + 1 lines. The first line should be "Case #x:" where x is the case number (starting from 1). The next **N** lines should contain the RPI of each team, one per line, in the same order as the schedule.

Answers with a relative or absolute error of at most 10^{-6} will be considered

correct.

Limits

1 ≤ **T** ≤ 20.
If the schedule contains a '1' in row i, column j, then it contains a '0' in row j, column i.
If the schedule contains a '0' in row i, column j, then it contains a '1' in row j, column i.
If the schedule contains a '.' in row i, column j, then it contains a '.' in row j, column i.
Every team plays at least two other teams.
No two teams play each other twice.
No team plays itself.

Small dataset

3 ≤ **N** ≤ 10.

Large dataset

3 ≤ **N** ≤ 100.

Sample

Input	Output
2 3 .10 0.1 10. 4 .11. 0.00 01.1 .10.	Case #1: 0.5 0.5 0.5 Case #2: 0.645833333333 0.368055555556 0.604166666667 0.395833333333

Problem B. Revenge of the Hot Dogs

Small input
15 points

Solve B-small

Large input
20 points

Solve B-large

Problem

Last year, several hot dog vendors were lined up along a street, and they had a tricky algorithm to spread themselves out. Unfortunately, the algorithm was very slow and they are still going. All is not lost though! The hot dog vendors have a plan: time to try a new algorithm!

The problem is that multiple vendors might be selling too close to each other, and then they will take each other's business. The vendors can move along the street at 1 meter/second. To avoid interfering with each other, they want to stand so that every pair of them is separated by a distance of at least **D** meters.

Remember that the street is really long, so there is no danger of running out of space to move in either direction. Given the starting positions of all hot dog vendors, you should find the minimum time they need before all the vendors are separated (each two vendors are at least **D** meters apart from each other).

Input

Each point of the street is labeled with a number, positive, negative or zero. A point labeled *p* is */p/* meters east of the point labeled *0* if *p* is positive, and */p/* meters west of the point labeled *0* if *p* is negative. We will use this labeling system to describe the positions of the vendors in the input file.

The first line of the input file contains the number of cases, **T**. **T** test cases follow. Each case begins with a line containing the number of points **C** that have at least one hot dog vendor in the starting configuration and an integer **D** -- the minimum distance they want to spread out to. The next **C** lines each contain a pair of space-separated integers **P**, **V**, indicating that there are **V** vendors at the point labeled **P**.

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 amount of time it will take for the vendors to spread out apart on the street. Answers with relative or absolute error of at most 10^{-6} will be accepted.

Limits

$1 \le \mathbf{T} \le 50$.
All the values **P** are integers in the range $[-10^5, 10^5]$.
Within each test case all **P** values are distinct and given in an increasing order.
The limit on the sum of **V** values is listed below. All the **V** values are positive integers.

Small dataset

$1 \le \mathbf{D} \le 5$
$1 \le \mathbf{C} \le 20$.
The sum of all the **V** values in one test case does not exceed 100.

Large dataset

$1 \le \mathbf{D} \le 10^6$
$1 \le \mathbf{C} \le 200$.
The sum of all **V** values does not exceed 10^6

Sample

Input	Output
2 3 2 0 1 3 2 6 1 2 2 0 3 1 1	Case #1: 1.0 Case #2: 2.5

Round 1B 2011

A. RPI

B. Revenge of the Hot Dogs

C. House of Kittens

Contest Analysis

Questions asked 1

Problem C. House of Kittens

Small input 20 points	Solve C-small
Large input 25 points	Solve C-large

Problem

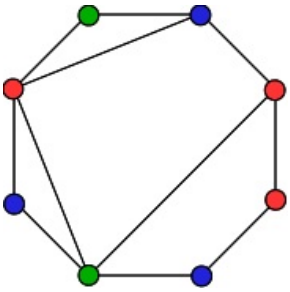
You have recently adopted some kittens, and now you want to make a house for them. On the outside, the house will be shaped like a convex polygon with **N** vertices. On the inside, it will be divided into several rooms by **M** interior walls connecting vertices along straight lines. No two walls will ever cross, but there might be multiple walls touching a single vertex.

So why is your house of kittens going to be so special? At every vertex, you are going to build a pillar entirely out of catnip! Kittens will be able to play with any pillar that touches the room they are in, giving them a true luxury home.

To make the house even more exciting, you want to use different flavors of catnip. A single pillar can only use one flavor, but different pillars can use different flavors. There is only one problem. If some room does not have access to *all* the catnip flavors in the house, then the kittens in that room will feel left out and be sad.

Your task is to choose what flavor of catnip to use for each vertex in such a way that (a) every flavor is accessible from every room, and (b) as many flavors as possible are used.

In the following example, three different flavors (represented by red, green, and blue dots) are distributed across an 8-sided house while keeping the kittens in every room happy:



In the image above, starting at the left corner of the top wall and going clockwise, the colors here are: green, blue, red, red, blue, green, blue, red.

Input

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

Each test case consists of three lines. The first line gives **N** and **M**, the number of vertices and interior walls in your cat house. The second lines gives space-separated integers $\mathbf{U}_1$, $\mathbf{U}_2$, ..., $\mathbf{U}_\mathbf{M}$ describing where each interior wall begins. The third lines gives space-separated integers $\mathbf{V}_1$, $\mathbf{V}_2$, ..., $\mathbf{V}_\mathbf{M}$ describing where each interior wall ends.

More precisely, if the vertices of your cat house are labeled 1, 2, ..., **N** in clockwise order, then the interior walls are between vertices $\mathbf{U}_1$ and $\mathbf{V}_1$, $\mathbf{U}_2$ and $\mathbf{V}_2$, etc.

Output

For each test case, output two lines. The first should be "Case #x: C", where x is the case number, and C is the maximum number of catnip flavors that can be used. The second line should contain **N** space-separated integers: "y_1 y_2 ... $y_\mathbf{N}$", where y_i is an integer between 1 and C indicating which catnip flavor you assigned to vertex i.

If there are multiple assignments with C flavors, you may output any of them.

Limits

$1 \le \mathbf{T} \le 100$.
$1 \le \mathbf{M} \le \mathbf{N} - 3$.
$1 \le \mathbf{U}_i < \mathbf{V}_i \le \mathbf{N}$ for all i.
Interior walls do not touch each other except at the **N** vertices.
Interior walls do not touch the outside of the house except at the **N** vertices.

Small dataset

4 ≤ **N** ≤ 8.

Large dataset

4 ≤ **N** ≤ 2000.

Sample

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

Problem A. Square Tiles

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

Problem

You are selling beautiful geometric pictures. Each one consists of 1x1 square tiles arranged into a non-overlapping grid. For example:

```
.##..
.####
.####
.##..
```

Blue tiles are represented by '#' characters, and white tiles are represented by '.' characters. You do not use other colors.

Not everybody likes blue though, and some customers want you to replace all the blue tiles in your picture with red tiles. Unfortunately, red tiles only come in the larger 2x2 size, which makes this tricky.

You can cover any 2x2 square of blue tiles with a single red tile, and then repeat until finished. A red tile cannot overlap another red tile, it cannot cover white tiles, and it cannot go outside the picture. For example, you could add red tiles to the previous picture as follows:

```
./\..
.\//\
./\\/
.\/..
```

Each red tile is represented here by a pair of '/' characters in the top-left and bottom-right corners, and a pair of '\' characters in the other two corners.

Given a blue and white picture, can you transform it into a red and white picture in this way?

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 **R** and **C**, the number of rows and columns in a picture. The next **R** lines each contain exactly **C** characters, describing the picture. As above, '#' characters represent blue tiles, and '.' characters represent white tiles.

Output

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

If it is possible to cover the blue tiles with non-overlapping red tiles, output **R** lines each containing **C** characters, describing the resulting red and white picture. As above, red tiles should be represented by '/' and '\' characters, while white tiles are represented by '.' characters. If multiple solutions are possible, you may output any of them.

If the task is impossible, output a single line containing the text "Impossible" instead.

Limits

Small dataset

$1 \le \mathbf{T} \le 20$.
$1 \le \mathbf{R} \le 6$.
$1 \le \mathbf{C} \le 6$.

Large dataset

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

Sample

Input	Output
3 2 3 ### ### 1 1 . 4 5 .##.. .#### .#### .##..	Case #1: Impossible Case #2: . Case #3: ./\.. .\//\ ./\\/ .\/..

Round 1C 2011

A. Square Tiles
B. Space Emergency
C. Perfect Harmony

Contest Analysis
Questions asked 2

Submissions

Square Tiles
10pt | Not attempted 4043/4140 users correct (98%)
10pt | Not attempted 3857/4035 users correct (96%)

Space Emergency
12pt | Not attempted 1442/2158 users correct (67%)
25pt | Not attempted 656/1158 users correct (57%)

Perfect Harmony
8pt | Not attempted 2839/3507 users correct (81%)
35pt | Not attempted 60/1308 users correct (5%)

Top Scores

Burunduk1	100
mystic	100
yuhch123	100
Qifeng.Chen	100
ikatanic	100
Smylic	100
Copludrm	100
AS1	100
zhendongjia	100
Akim	100

Problem B. Space Emergency

Small input
12 points
Solve B-small

Large input
25 points
Solve B-large

Problem

There's an emergency—in space! You need to send your fleet's flagship as quickly as possible from star 0 to star **N**, traveling through the other stars in increasing numerical order along the way (0→1→...→**N**). Your flagship normally travels at a speed of 0.5 parsecs per hour.

In addition to sending your flagship, you can order your engineers to build up to **L** speed boosters at different stars. Building a speed booster takes **t** hours, and all **L** speed boosters can be built in parallel. While your flagship travels from a star with a completed speed booster to the next star, its speed is 1 parsec per hour.

If a speed booster is completed at a star while your flagship is traveling from that star to the next one, your flagship will start moving faster as soon as the speed booster is completed.

How many hours does it take your flagship to get to star **N** if you build speed boosters to make it arrive as soon as possible?

Input

The first line of the input gives the number of test cases, **T**. **T** lines follow. Each contains integers, **L**, **t**, **N** and **C**, followed by **C** integers $\mathbf{a_i}$, all separated by spaces. $\mathbf{a_i}$ is the number of parsecs between star **k*C**+i and star **k*C**+i+1, for all integer values of **k**.

For example, with **N**=8, **C**=3, $\mathbf{a_0}$=3, $\mathbf{a_1}$=5 and $\mathbf{a_2}$=4, the distances between stars are [3, 5, 4, 3, 5, 4, 3, 5].

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 single integer: the number of hours it takes to reach star **N**. The answer is guaranteed to always be an integer.

Limits

$1 \le \mathbf{T} \le 100$.
$1 \le \mathbf{C} \le 1000$.
$\mathbf{C} \le \mathbf{N}$.
$1 \le \mathbf{a_i} \le 10^4$.
$0 \le \mathbf{t} \le 10^{11}$.
t is even.

Small dataset

$1 \le \mathbf{N} \le 1000$.
$0 \le \mathbf{L} \le 2$.

Large dataset

$1 \le \mathbf{N} \le 10^6$.
$0 \le \mathbf{L} \le N$.

Sample

Input	Output
2 2 20 8 2 3 5 1 4 2 2 10 4	Case #1: 54 Case #2: 20

Explanation

In the second case, we can build one speed booster. The distances between stars are [10, 4]. We build the speed booster on the first star. After 4 hours, our flagship has gone 2 parsecs and the speed booster is complete. It takes our

flagship another 8 hours to get to star 1, then 8 more hours to get to star 2, our destination.

Note: This problem takes place in a universe where the speed of light is much higher than 1 parsec per hour, so we don't have to worry about special relativistic effects.

Round 1C 2011

A. Square Tiles
B. Space Emergency
C. Perfect Harmony

Contest Analysis
Questions asked 2

Submissions

Square Tiles
10pt | Not attempted **4043/4140 users** correct (98%)
10pt | Not attempted **3857/4035 users** correct (96%)

Space Emergency
12pt | Not attempted **1442/2158 users** correct (67%)
25pt | Not attempted **656/1158 users** correct (57%)

Perfect Harmony
8pt | Not attempted **2839/3507 users** correct (81%)
35pt | Not attempted **60/1308 users** correct (5%)

Top Scores

Burunduk1	100
mystic	100
yuhch123	100
Qifeng.Chen	100
ikatanic	100
Smylic	100
Copludrm	100
AS1	100
zhendongjia	100
Akim	100

Problem C. Perfect Harmony

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

Small input 8 points	Solve C-small
Large input 35 points	Solve C-large

Problem

Jeff is a part of the great Atlantean orchestra. Each player of the orchestra has already decided what sound will he play (for the sake of simplicity we assume each player plays only one sound). We say two sounds are in harmony if the frequency of any one of them divides the frequency of the other (that's a pretty restrictive idea of harmony, but the Atlanteans are known to be very conservative in music). Jeff knows that the notes played by other players are not necessarily in harmony with each other. He wants his own note to improve the symphony, so he wants to choose his note so that it is in harmony with the notes all the other players play.

Now, this sounds simple (as all the frequencies are positive integers, it would be enough for Jeff to play the note with frequency 1, or, from the other side, the Least Common Multiple of all the other notes), but unfortunately Jeff's instrument has only a limited range of notes available. Help Jeff find out if playing a note harmonious with all others is possible.

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 numbers: **N**, **L** and **H**, denoting the number of other players, the lowest and the highest note Jeff's instrument can play. The second line contains **N** integers denoting the frequencies of notes played by the other players.

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 string "NO" (if Jeff cannot play an appropriate note), or a possible frequency. If there are multiple frequencies Jeff could play, output the lowest one.

Limits

$1 \le \mathbf{T} \le 40$.

Small dataset

$1 \le \mathbf{N} \le 100$.
$1 \le L \le H \le 10000$.
All the frequencies are no larger than 10000.

Large dataset

$1 \le \mathbf{N} \le 10^4$.
$1 \le \mathbf{L} \le \mathbf{H} \le 10^{16}$
All the frequencies are no larger than 10^{16}

Sample

Input	Output
2 3 2 100 3 5 7 4 8 16 1 20 5 2	Case #1: NO Case #2: 10

Powered by

Google Cloud Platform

Problem A. Airport Walkways

Small input 8 points	Solve A-small
Large input 10 points	Solve A-large

Problem

You're in an airport, standing at point 0. A corridor of length **X** leads to the gate, where your plane is about to leave. There are moving walkways in the corridor, each moving with some speed $\mathbf{w_i}$. When you walk or run on one of those, you move with speed (your speed + $\mathbf{w_i}$). The walkways do not change their position; they just make you move faster. The walkways do not overlap: at any given point of the corridor there is at most one walkway, but one walkway can begin at the point where another ends.

Your normal walking speed is **S**. You are worried that you might not catch your plane, though, so you can run a bit - you can run with speed **R** for at most **t** seconds in total. You do not have to run for **t** consecutive seconds: you can split these **t** seconds into any number of intervals, or even not use some part of them.

How long does it take you to get to the gate, assuming you choose when to walk and when to run in order to reach it as soon as possible?

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 five integers: **X** (the length of the corridor, in meters), **S** (your walking speed, in meters per second), **R** (your running speed, in meters per second), **t** (the maximum time you can run, in seconds) and **N** (the number of walkways).

Each of the next **N** lines contains three integers: $\mathbf{B_i}$, $\mathbf{E_i}$ and $\mathbf{w_i}$ - the beginning and end of the walkway (in meters from your starting point) and the speed of the walkway (in meters per second).

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 time (in seconds) you need to reach point **X** if you walk and run optimally. Answers with relative or absolute error of at most 10^{-6} will be accepted.

Limits

$1 \le \mathbf{T} \le 40$.
$1 \le \mathbf{S} < \mathbf{R} \le 100$.
$1 \le \mathbf{w_i} \le 100$.
$0 \le \mathbf{B_i} < \mathbf{E_i} \le \mathbf{X}$.
$\mathbf{E_i} \le \mathbf{B_{i+1}}$.

Small dataset

$1 \le \mathbf{t} \le 100$.
$1 \le \mathbf{X} \le 100$.
$1 \le \mathbf{N} \le 20$.

Large dataset

$1 \le \mathbf{t} \le 10^6$.
$1 \le \mathbf{X} \le 10^6$.
$1 \le \mathbf{N} \le 1000$.

Sample

Input	Output
3 10 1 4 1 2 4 6 1 6 9 2 12 1 2 4 1 6 12 1 20 1 3 20 5 0 4 5	Case #1: 4.000000 Case #2: 5.500000 Case #3: 3.538095238

```
4 8 4
8 12 3
12 16 2
16 20 1
```

The best solution in the first case is to start running immediately and run for one second.

Round 2 2011

A. Airport Walkways
B. Spinning Blade
C. Expensive Dinner
D. A.I. War

Contest Analysis
Questions asked

Problem B. Spinning Blade

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

Problem

Being bored with the traps in your secret hideout design, you decided to go for something classical, but always enjoyable - the *spinning blade*. You ordered a really heavy metal sheet out of which you will cut the blade; a uniform square **C**-by-**R** grid will be painted on the sheet. You have determined the best shape for the blade -- you will first cut a large square consisting of **K**-by-**K** grid cells, where **K** ≥ 3. Then, you will cut out the four 1-by-1 corner cells out of the square to end up with a *blade*. After determining all this, you started waiting for the sheet to arrive.

When the sheet arrived, you were shocked to find out that the sheet had imperfections in it! You expected each cell to have mass **D**, but it turned out that the mass can vary a bit because of differences in thickness. This is bad because you want to insert a shaft exactly in the center of the blade and spin it very fast, so the center of mass of the blade must be exactly in its center as well. The definition of the center of mass of a flat body can be found below.

Given the grid and the mass of each cell, what is the largest possible size of the blade you can make so that the center of mass is exactly in its center?

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 3 integers: **R**, **C** and **D** — the dimensions of the grid and the mass you expected each cell to have. The next **R** lines each contain **C** digits $\mathbf{w_{ij}}$ each, giving the differences between the actual and the expected mass of the grid cells. Each cell has a uniform density, but could have an integer mass between **D + 0** and **D + 9**, inclusive.

Output

For each test case, output one line containing "Case #x: **K**", where x is the case number (starting from 1) and **K** is the largest possible size of the blade you can cut out. If no acceptable blade of size at least 3 can be found, print "IMPOSSIBLE" instead.

Limits

$1 \le \mathbf{T} \le 20$.
$0 \le \mathbf{w_{ij}} \le 9$.
The size of the input file will not exceed 625KB.

Small dataset

$3 \le \mathbf{R} \le 10$.
$3 \le \mathbf{C} \le 10$.
$1 \le \mathbf{D} \le 100$.

Large dataset

$3 \le \mathbf{R} \le 500$.
$3 \le \mathbf{C} \le 500$.
$1 \le \mathbf{D} \le 10^6$.

Sample

Input	Output
2 6 7 2 1111111 1122271 1211521 1329131 1242121 1122211 3 3 7 123 234 345	Case #1: 5 Case #2: IMPOSSIBLE

Note

The *center of mass* of a 2D object is formally defined as a point **c**. If you compute the sum of (**p** - **c**) * mass(**p**) for all points **p** in the object, you must get **0**. Here, **p**, **c** and **0** are two-dimensional vectors. This definition also works if you treat each grid cell as a "point", with all of its mass at its center.

In real life, you could place your finger under a flat object's center of mass, and balance that object on your finger. It would not fall.

To illustrate with an example, the only blade that is possible to cut out in the second sample test case, the 3x3 blade created by cutting away the corners, has its center of mass at the point (1.54, 1.46), where we assume the bottom-left corner of the sheet has coordinates (0, 0), and the coordinates grow right and up, respectively. This is verified by checking the following equality:
(-1.04, 0.04) * 9 + (-0.04, 1.04) * 9 + (-0.04, 0.04) * 10 + (-0.04, -0.96) * 11 + (0.96, 0.04) * 11 = (0, 0).

Problem C. Expensive Dinner

Small input 13 points	Solve C-small
Large input 17 points	Solve C-large

Problem

Your friends are all going to a restaurant for dinner tonight. They're all very good at math, but they're all very strange: your $\mathbf{a}^{th}$ friend (starting from 1) will be *unhappy* unless the total cost of the meal is a positive integer, and is divisible by **a**.

Your friends enter the restaurant one at a time. As soon as someone enters the restaurant, if that person is unhappy then the group will *call* a waiter immediately.

As long as there is at least one unhappy person in the restaurant, one of those unhappy people will buy the lowest-cost item that will make him or her happy. This will continue until nobody in the restaurant is unhappy, and then the waiter will leave. Fortunately, the restaurant sells food at every integer price. See the explanation of the first test case for an example.

Your friends could choose to enter the restaurant in any order. After the waiter has been called, if there is more than one unhappy person in the restaurant, any one of those unhappy people could choose to buy something first. The way in which all of those choices are made could have an effect on how many times the group calls a waiter.

As the owner of the restaurant, you employ some very tired waiters. You want to calculate the **spread** of your friends: the difference between the maximum number of times they might call a waiter and the minimum number of times they might call a waiter.

Input

The first line of the input gives the number of test cases, **T**. **T** test cases follow, each on its own line. Each test case will contain one integer **N**, the number of friends you have.

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 spread for that test case.

Limits

Small dataset

$1 \le \mathbf{T} \le 100$.
$1 \le \mathbf{N} \le 1000$.

Large dataset

$1 \le \mathbf{T} \le 1000$.
$1 \le \mathbf{N} \le 10^{12}$.

Sample

Input	Output
4 1 3 6 16	Case #1: 0 Case #2: 1 Case #3: 2 Case #4: 5

Explanation

In Case #2, suppose your friends arrive in the order [1, 2, 3]. Then #1 arrives; is unhappy; calls a waiter; and buys something costing 1. Now nobody is unhappy. #2 arrives next; is unhappy; calls a waiter; and buys something costing 1 (for a total of 2). Now nobody is unhappy. #3 arrives next; is unhappy; calls a waiter; and buys something costing 1 (for a total of 3). Now

#2 is unhappy, and buys something costing 1 (for a total of 4). Now #3 is unhappy, and buys something costing 2 (for a total of 6). Finally nobody is unhappy, and a waiter was called three times.

Suppose instead that your friends arrived in the order [3, 1, 2]. Then #3 arrives; is unhappy; calls a waiter; and buys something costing 3. Now nobody is unhappy. #1 arrives next; nobody is unhappy. #2 arrives next; is unhappy; calls a waiter; and buys something costing 1 (for a total of 4). Now #3 is unhappy, and buys something costing 2 (for a total of 6). Now nobody is unhappy, and a waiter was called two times. The spread is 1.

Problem D. A.I. War

Small input 10 points	Solve D-small
Large input 22 points	Solve D-large

Introduction

A.I. War is a real-time strategy game developed by Arcen Games. This problem was inspired by the game, but does not assume you have played it.

Problem

You're facing an artificial intelligence in a deadly war for the future of the galaxy. In order to defeat the A.I., you will need to threaten its *home planet*. Some planets are connected to each other by wormholes; any planet may be connected to any number of other planets using the wormholes.

You begin by owning only your home planet. Each turn, you may conquer any planet you *threaten*. You threaten a planet if you don't own it, and it is connected by a wormhole to any of the planets you own. Once you have conquered a planet, you own it. As soon as you threaten the A.I.'s home planet, you may not conquer any more planets.

While attending the most important day in tactical school, you discovered two things about the A.I.:

- For each planet you conquer, the A.I. will become more powerful, because it will see you as a threat and produce more ships to defend itself.
- The A.I. will defend every planet you're currently threatening.

You have combined those two facts to create a strategy:

1. You will conquer planets until you threaten the A.I.'s home base.
2. If there are multiple ways of completing step 1, do it while conquering the **smallest** possible number of planets.
3. If there are multiple ways of completing step 2, do it so that at the end you will threaten the **largest** possible number of planets.

Given the planets and the wormholes, how many planets will you conquer and threaten on your way to the A.I.'s home base if you follow the strategy described above?

Input

The first line of the input gives the number of test cases, **T**. **T** test cases follow. Each test case starts with a single line containing two space-separated integers: **P**, the number of planets, and **W**, the number of wormholes. Your home planet is planet 0, and the A.I.'s home planet is planet 1.

The second line of each test case will contain **W** space-separated pairs of comma-separated integers $\mathbf{x}_i$,$\mathbf{y}_i$. Each of these indicates that there is a two-way wormhole connecting planets $\mathbf{x}_i$ and $\mathbf{y}_i$.

Output

For each test case, output one line containing "Case #x: c t", where x is the case number (starting from 1), c is the number of planets you conquer if you follow the above strategy, and t is the number of planets you threaten at the end (including the A.I.'s home planet).

Limits

$1 \le \mathbf{T} \le 50$.
$0 \le \mathbf{x}_i < \mathbf{y}_i < \mathbf{P}$.
Each wormhole is unique: If $i \ne j$, then $(\mathbf{x}_i, \mathbf{y}_i) \ne (\mathbf{x}_j, \mathbf{y}_j)$.
There will be at least one way to reach the A.I.'s home planet from your home planet using a series of wormholes.

Small dataset

$2 \le \mathbf{P} \le 36$.
$1 \le \mathbf{W} \le 630$.

Large dataset

$2 \le \mathbf{P} \le 400$.

$1 \leq \mathbf{W} \leq 2000$.

Sample

Input	Output
4 2 1 0,1 3 3 0,1 1,2 0,2 5 5 0,4 0,2 2,4 1,2 1,4 7 9 0,6 0,2 0,4 2,4 3,4 2,3 3,5 4,5 1,5	Case #1: 0 1 Case #2: 0 2 Case #3: 1 2 Case #4: 2 4

Explanation

In the first case, you don't have to conquer anything, and you're already threatening the A.I.'s home planet.

In the third case, you can threaten the A.I.'s home planet after conquering only one planet. You end up threatening two planets, and there's an extra planet that isn't connected to anything.

In the fourth case, you can threaten the A.I.'s home planet by conquering planets 4 and 5. You end up threatening planets 6, 2, 3 and 1 (the A.I.'s home planet).

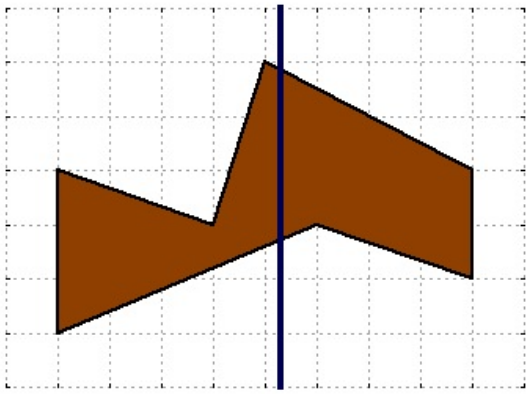
Note

Arcen Games is the creator of A.I. War. Arcen Games does not endorse and has no involvement with Google Code Jam.

Problem A. Irregular Cakes

Problem

Mary the Mathematician has a bakery that she founded some years ago, but after all this time she has become bored with always baking the same rectangular and circular cakes. For her next birthday, she wants to bake an *irregular* cake, which is defined as the area between two "polylines" between x=0 and x=**W**. These polylines will be called the lower boundary and the upper boundary.



Formally, a polyline is defined by a sequence of points (P_0, P_1, ..., P_n) going from left to right. Consecutive points are connected to form a sequence of line segments, which together make up the polyline.

Today is Mary's birthday and she has baked an irregular cake bounded by two polylines with **L** points and **U** points respectively. After singing "Happy Birthday," she wants to make **G**-1 vertical cuts to split the cake into **G** slices with equal area. She can then share these cake slices with all her guests. However, the irregular cake shape makes this task pretty tricky. Can you help her decide where to make the cuts?

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 four integers: **W** (the cake's width), **L** (the number of points on the lower boundary), **U** (the number of points on the upper boundary) and **G** (the number of guests at the party).

This is followed by **L** lines specifying the lower boundary. The i-th line contains two integers x_i and y_i, representing the coordinates of the i-th point on the lower boundary. This is followed by **U** more lines specifying the upper boundary. The j-th line here contains two integers x_j and y_j, representing the coordinates of the j-th point on the upper boundary.

Output

For each test case, output **G** lines. The first line should be "Case #x:" where x is the case number (starting from 1). The next **G**-1 lines should contain the x-coordinates at which cuts must be made, ordered from the leftmost cut to the rightmost cut.

Answers with a relative or absolute error of at most 10^{-6} will be considered correct.

Limits

$1 \le \mathbf{T} \le 100$.
$1 \le \mathbf{W} \le 1000$.
$2 \le \mathbf{L} \le 100$.
$2 \le \mathbf{U} \le 100$.
All coordinates will be integers between -1000 and 1000, inclusive.
The x-coordinate of the leftmost point of both boundaries will be 0.
The x-coordinate of the rightmost point of both boundaries will be **W**.
Points in the same boundary will be sorted increasingly by x-coordinate.
Points in the same boundary will have different x-coordinates.

The lower boundary will always be strictly below the upper boundary for all x between 0 and **W**, inclusive. (In other words, the lower boundary will have a smaller y-coordinate than the upper boundary at every x position.)

Small dataset

$2 \leq \mathbf{G} \leq 3$.

Large dataset

$2 \leq \mathbf{G} \leq 101$.

Sample

Input	Output
2 15 3 3 3 0 6 10 8 15 9 0 10 5 11 15 13 8 3 4 2 0 2 5 4 8 3 0 5 3 4 4 7 8 5	Case #1: 5.000000 10.000000 Case #2: 4.290588

Problem B. Dire Straights

Small input
4 points

Solve B-small

Large input
12 points

Solve B-large

Problem

You are playing a card game, where each card has an integer number written on it.

To play the game, you are given some cards — your *hand*. Then you arrange the cards in your hand into *straights*. A straight is a set of cards with consecutive values; e.g. the three cards {3, 4, 5}, or the single card {7}. You then receive a number of dollars equal to the length of the shortest straight. If you have no cards, you can form no straights, so you get zero dollars.

You will be given a series of test cases, each of which describes the cards you will have in your hand. Find the maximum number of dollars you can receive for each test case.

Input

The first line of the input contains the number of test cases, **T**. Each test case consists of one line. Each line contains **N**, the number of cards in your hand, followed by **N** integers giving the numbers on those cards. These numbers are all space-separated.

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 dollars you can receive.

Limits

$1 \le \mathbf{T} \le 100$
The numbers on the cards are between 1 and 10000.

Small dataset

$0 \le \mathbf{N} \le 10$

Large dataset

$0 \le \mathbf{N} \le 1000$

Sample

Input	Output
4 10 1 2 3 4 5 10 9 8 7 6 8 101 102 103 104 105 106 103 104 0 5 1 2 3 4 9	Case #1: 10 Case #2: 4 Case #3: 0 Case #4: 1

In case 1, you have ten cards numbered 1 to 10, so you make one straight of length 10, and get 10 dollars.

In case 2, you could make two straights {101,102,103,104,105,106} and {103,104} and get 2 dollars. But it would be better to make {101,102,103,104} and {103,104,105,106} and get 4 dollars.

In case 4, the card with the number 9 must be in a straight containing only that card. So you get 1 dollar.

In case 3, you have zero cards, so you get zero dollars. You don't get money for nothing.

Powered by

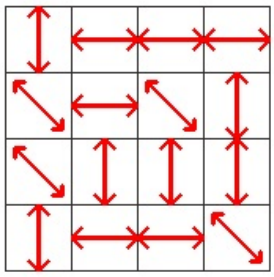
Google Cloud Platform

Problem C. Perpetual Motion

Small input 5 points	Solve C-small
Large input 24 points	Solve C-large

Problem

Have you ever been to the Google Lemming Factory? It is a very unusual place. The floor is arranged into an **R** x **C** grid. Within each grid square, there is a conveyor belt oriented up-down, left-right, or along one of the two diagonals. The conveyor belts move either forwards or backwards along their orientations, and you can independently choose which of the two possible directions each conveyor belt should move in.

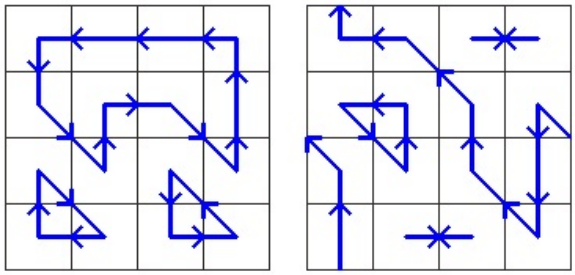


Currently, there is a single lemming standing at the center of each square. When you start the conveyor belts, each lemming will move in the direction of the conveyor belt he is on until he reaches the center of a new square. All these movements happen simultaneously and take exactly one second to complete. Afterwards, the lemmings will all be on new squares, and the process will repeat from their new positions. This continues forever, or at least until you turn off the conveyor belts.

- When a lemming enters a new square, he continues going in the direction he was already going until he reaches the center of that square. He will not be affected by the new conveyor belt until the next second starts.
- If a lemming moves off the edge of the grid, he comes back at the same position on the opposite side. For example, if he were to move diagonally up and left from the top-left square, he would arrive at the bottom-right square. By the miracle of science, this whole process still only takes 1 second.
- Lemmings never collide and can always move past each other without difficulty.

The trick is to choose directions for each conveyor belt so that the lemmings will keep moving forever without ever having two of them end up in the center of the same square at the same time. If that happened, they would be stuck together from then on, and that is not as fun for them.

Here are two ways of assigning directions to each conveyor belt from the earlier example:



In both cases, we avoid ever sending two lemmings to the center of the same square at the same time.

Given an arbitrary floor layout, calculate **N**, the number of ways to choose directions for each conveyor belt so that no two lemmings will ever end up in the center of the same square at the same time. The answer might be quite large, so please output it modulo 1000003.

Input

The first line of input gives the number of test cases, **T**. **T** test cases follow. Each begins with a line containing positive integers **R** and **C**.

This is followed by **R** lines, each containing a string of **C** characters chosen from "|-/\". Each character represents the orientation of the conveyor belt in a single square:

- '|' represents a conveyor belt that can move up or down.
- '-' represents a conveyor belt that can move left or right.
- '/' represents a conveyor belt that can move up-right or down-left.
- '\' represents a conveyor belt that can move up-left or down-right.

Output

For each test case, output one line containing "Case #x: **M**", where x is the case number (starting from 1), and **M** is the remainder when dividing **N** by 1000003.

Limits

1 ≤ **T** ≤ 25.

Small dataset

3 ≤ **R** ≤ 4.
3 ≤ **C** ≤ 4.

Large dataset

3 ≤ **R** ≤ 100.
3 ≤ **C** ≤ 100.

Sample

Input	Output
3 3 3 |-/ ||| --| 3 4 ---- |||| \\// 4 4 |--- \-| \\\|| |--\	Case #1: 2 Case #2: 0 Case #3: 16

Problem D. Mystery Square

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 D-small
Large input 31 points	Solve D-large

Problem

I have written down a large perfect square in binary, and then replaced some of the digits with question marks. Can you figure out what my original number was?

Input

The first line of the input gives the number of test cases, **T**. **T** test cases follow, one per line. Each line contains **S**: a perfect square written in binary, but with some of the digits replaced by question marks.

Output

For each test case, output one line containing "Case #x: **N**", where x is the case number (starting from 1) and **N** is a perfect square written in binary, obtained by replacing each '?' character in **S** with either a '0' character or a '1' character.

Limits

1 ≤ **T** ≤ 25.
S begins with '1'.
S contains only the characters '0', '1', and '?'.
In every test case, there is exactly one possible choice for **N**.

Small dataset

S is at most 60 characters long.
S contains at most 20 '?' characters.

Large dataset

S is at most 125 characters long.
S contains at most 40 '?' characters.

Sample

Input	Output
3 1??? 1 10??110??00??1000??	Case #1: 1001 Case #2: 1 Case #3: 10111101100001000 01

Problem A. Runs

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 14 points	Solve A-small
Large input 16 points	Solve A-large

Problem

I have a string **S** consisting of lower-case alphabetic characters, 'a' - 'z'. Each maximal sequence of contiguous characters that are the same is called a "run". For example, "bookkeeper" has 7 runs. How many different permutations of **S** have exactly the same number of runs as **S**?

Two permutations a and b are considered different if there exists some index `i` at which they have a different character: `a[i] ≠ b[i]`.

Input

The first line of the input gives the number of test cases, **T**. **T** lines follow. Each contains a single non-empty string of lower-case alphabetic characters, **S**, the string of interest.

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 different permutations of **S** that have exactly the same number of runs as **S**, modulo 1000003.

Limits

$1 \le \mathbf{T} \le 100$.
S is at least 1 character long.

Small dataset

S is at most 100 characters long.

Large dataset

S is at most 450000 characters long.
S has at most 100 runs.
The input file will not exceed 1 megabyte in size.

Sample

Input	Output
2 aabcd bookkeeper	Case #1: 24 Case #2: 7200

World Finals 2011

A. Runs
B. Rains Over Atlantis
C. Program within a Program
D. Ace in the Hole
E. Google Royale

Contest Analysis
Questions asked 1

Submissions

Runs
14pt | Not attempted 11/12 users correct (92%)
16pt | Not attempted 10/11 users correct (91%)

Rains Over Atlantis
7pt | Not attempted 17/18 users correct (94%)
23pt | Not attempted 3/7 users correct (43%)

Program within a Program
15pt | Not attempted 1/1 users correct (100%)
23pt | Not attempted

Ace in the Hole
20pt | Not attempted 8/14 users correct (57%)
22pt | Not attempted

Google Royale
20pt | Not attempted 20/22 users correct (91%)
40pt | Not attempted 1/1 users correct (100%)

Top Scores

rng..58	90
mystic	77
meret	77
RAD.	77
misof	70
g201513	60
pashka	57
vepifanov	57
eatmore	50
winger	50

Problem B. Rains Over Atlantis

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 B-small
Large input 23 points	Solve B-large

Problem

Rains fall on the isle of Atlantis, and will erode all the land to nothingness. What you want to know, so that you can organize the evacuation, is how soon it will happen.

You have a map of Atlantis. The map is a square grid, and each square contains the height of the land in that square, in metres, above sea level. All squares outside the map have height 0; all squares with height 0 are water, and all squares with larger heights are land. There are no squares with lower height.

Water can *flow* from a source square to a target square if the source square and target square share an edge, and if the height of the water in the target square is lower than or equal to the height of water in the source square.

It's raining very quickly, which means that if there is nowhere for the rain water in a square to flow, water in that square will accumulate until there is a square to which the rain water can flow. Squares that are not on the map can accept any amount of flow. For example, the following map:

```
5 9 9 9 9 9
0 8 9 0 2 5
3 9 9 9 9 9
```

Will quickly fill up with water. We'll call the height of water in each square, plus the height of the land, the *water level*. It will be:

```
5 9 9 9 9 9
0 8 9 5 5 5
3 9 9 9 9 9
```

Note that the 0 in the middle of the land, although it's water, is not connected to the outside of the map and so just accumulates water. The 0 on the border of the land, however, is connected to the outside of the map, and so the water from the 8 can flow through it to the outside.

The direction in which water flows is determined by the water level. If there are multiple possible squares where water could flow from one particular source square, the water from that source will flow to the square with the lowest water level (ties don't matter, as you will see).

Now the erosion begins. Each day, a square is eroded—its height decreases—depending on how water is flowing from it. If water is flowing from S to T, then S's height decreases by `min(WaterLevel(S) - WaterLevel(T), M)`. All erosion happens at exactly the same time, at the end of the day. For example, with **M**=5, the map above will erode to:

```
0 4 4 4 4 4
0 3 5 0 2 0
0 4 4 4 4 4
```

After a day's erosion, excess water will flow away: squares with water level higher than a neighbour's water level will lose water until they are of the same height. Water will also accumulate in the same way that it did on the first day. After the first day, this map's water level will become:

```
0 4 4 4 4 4
0 3 5 2 2 0
0 4 4 4 4 4
```

After another day of erosion, the map will look like:

```
0 0 0 0 0 0
0 0 2 0 0 0
0 0 0 0 0 0
```

...and the Atlanteans will need to escape in a big hurry. Your task is to determine how many days it will take for all the heights to erode to 0.

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 three space-separated integers: **H**, **W** and **M**. The first two denote the size of the map, while the third is the maximum amount a square can erode in one day, as described above. **H** lines follow, each of which contains **W** space-separated integers. The i^{th} integer on the j^{th} line denotes the height of the square at (i, j).

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 days it takes to erode all the island.

Limits

$1 \leq \mathbf{T} \leq 40$.

Small dataset

$1 \leq \mathbf{H}, \mathbf{W} \leq 10$.
$1 \leq \mathbf{M} \leq 100$.
$0 \leq$ all heights ≤ 100.

Large dataset

$1 \leq \mathbf{H}, \mathbf{W} \leq 20$.
$1 \leq \mathbf{M} \leq 10^{15}$.
$0 \leq$ all heights $\leq 10^{15}$.

Sample

Input	Output
2 3 6 5 5 9 9 9 9 9 0 8 9 0 2 5 3 9 9 9 9 9 3 6 3 3 8 10 11 10 8 7 5 2 12 8 8 6 9 11 9 8 4	Case #1: 3 Case #2: 5

In the second case, the water height looks like:
3 8 10 11 10 8
7 **7** **7** 12 8 8
6 9 11 9 8 4

After one day the island looks as follows:
0 5 7 8 7 5
4 5 2 9 8 5
3 6 8 6 5 1

And after the second day:
0 2 4 5 4 2
1 4 2 6 5 2
0 3 5 3 2 0

And the third day:
0 0 1 2 1 0
0 1 2 3 2 0
0 0 2 0 0 0

After the fourth day, things are looking desperate for the Atlanteans:
0 0 0 0 0 0
0 0 1 0 0 0
0 0 0 0 0 0

Finally, on the fifth day the last square erodes away. Atlantis lasted for five days; they probably shouldn't have built their city out of brown sugar.


Powered by
Google Cloud Platform

Problem C. Program within a Program

Small input 15 points	Solve C-small
Large input 23 points	Solve C-large

Problem

You have a robot on an infinite east-west highway, and it has a cake to deliver. Every mile along the highway, in both directions, there is a lamppost. You want to program the robot to move exactly **N** lampposts to the east, and release the cake there. The route does not have to be direct, as long as the robot eventually releases the cake in the right place.

Unfortunately, the robot comes equipped with only very little memory, and it is capable of no advanced logic. To control the robot you will have to give it a very simple program at the start that will get it to release the cake at the proper location. This program must be composed of one or more statements, each of which tells the robot what to do under certain conditions. These statements must be in the following format:

```
<S> <M> -> <action>
```

which means that if all of the following conditions are met:

1. The robot is in state `S`.
2. The robot is at a lightpost marked with number `M`.

then it will perform exactly one of the following actions:

1. Mark the current post with a new number, change state and move. To do this `<action>` must be formatted as `"<D> <NS> <NM>"`, where `D` is the direction to move (use 'W' for west and 'E' for east), `NS` is the robot's new state and `NM` is the new mark for the current lightpost.
2. Release the cake at the current position and self-destruct. To do this `<action>` must be formatted as `"R"`.

If you output two or more statements with the same values of `S` and `M`, the robot will misbehave and destroy the cake.

If at any time the robot is in a state `X` at a lamppost marked with `Y` such that there is no statement with `S=X` and `M=Y`, then the robot will get confused and eat the cake.

All states and marks must be integers with absolute value no greater than one million (10^6). Assume that initially the robot is in state zero and all lampposts are marked with zero.

Given **N**, write a program so the robot releases the cake in the appropriate place. Your program must use at most 30 statements, and it must terminate within **X** steps.

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 an integer **N**, which indicates the lamppost where the robot must release the cake.

Output

For each test case, first output "Case #x: y", where x is the number of the test case (starting with 1) and y is the number of statements you will use. Next output y lines, each of which represents a statement for the robot in the format described previously.

WARNING: Judge's response might take up to 5 seconds longer than usual to appear, because your output is run as part of the validation.

Limits

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

Small dataset

$0 \le \mathbf{N} \le 500$.
$\mathbf{X} = 250{,}000$ (2.5×10^5).

Large dataset

$0 \le \mathbf{N} \le 5000$.
$\mathbf{X} = 150{,}000$ (1.5×10^5)

Sample

Input	Output
3 0 4 0	Case #1: 1 0 0 -> R Case #2: 5 0 0 -> E 1 1 1 0 -> E 2 1 2 0 -> E 3 1 3 0 -> E -1 1 -1 0 -> R Case #3: 3 0 0 -> E 1 1 0 1 -> R 1 0 -> W 0 1

In the first case, the robot is initially in state zero, and there is a zero on the lamppost. So it executes its only statement, which is to release the cake.

In the second case, the robot has five states: 0, 1, 2, 3, and -1. The robot performs the following actions:

- Mark the current lamppost with a 1, move east, and go to state 1.
- Mark the current lamppost with a 1, move east, and go to state 2.
- Mark the current lamppost with a 1, move east, and go to state 3.
- Mark the current lamppost with a 1, move east, and go to state -1.
- Release the cake.

In the third case, the robot has two states, and performs the following actions:

- Mark the current lamppost with a 1, move east, and go to state 1.
- Mark the current lamppost with a 1, move west, and go to state 0.
- Release the cake.

Note that the robot takes different actions at the two times it is in state 0, because it sees a different mark each time.

Problem D. Ace in the Hole

Small input 20 points	Solve D-small
Large input 22 points	Solve D-large

Problem

Amy has a deck of **N** cards with values 1 through **N**. She arranges the deck so that the values of the cards have no decreasing subsequence of length 3. For example, 1, 5, 4, 6, 3, 2 would be an illegal ordering because 5, 3, 2 is decreasing.

Amy now gives the deck of cards to Ben. Ben knows that the deck has no decreasing subsequence of length 3, but he does not know the exact ordering. He wants to find the card with value 1. He does this by choosing an arbitrary card, picking it up to observe its value, and then repeating until he has found the card with value 1. At each step, Ben chooses a card that will minimize the worst-case number of cards he has to examine.

Ben later tells you that he got unlucky and had to examine all **N** cards before finding the card with value 1. Given the order in which he examined the cards of the deck, what value did each card have? If there are multiple possibilities, choose the lexicographically greatest one.

A deck A is lexicographically greater than a deck B if and only if, at the first index at which they differ, the card in A has a value greater than the value of the card in B.

Example: **N** = 3, and Ben tried the cards in order 2, 1, 3 (the indices are 1-based). The values of the cards must have been: 2, 3, 1.

Explanation: If card #2 had value 1, then Ben would have stopped immediately. If card #2 had value 2, then Ben would have known the first card must have been the 1, because the ordering (3, 2, 1) is a decreasing subsequence of length 3, and thus could not have been the ordering. In either case, Ben would not have needed 3 guesses. Therefore, we can deduce card #2 have had value 3. Similarly, card #1 could not have had value 1, or Ben could have stopped early. Therefore, the card values must have been 2, 3, 1.

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 one integer **N**, the number of cards in the deck. The next line will contain **N** integers separated by single spaces, describing the order in which Ben examined the deck: the first integer denotes the 1-based position of the first card he examined, the second integer denotes the 1-based position of the second card he examined, and so on.

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 sequence of cards' values, separated by spaces.

Limits

$1 \le \mathbf{T} \le 100$
For the provided sequence of guesses, there will be at least one deck that meets all the constraints of the problem, including the constraint that Ben's strategy required him to look at **N** cards.

Small dataset

$1 \le \mathbf{N} \le 8$

Large dataset

$1 \le \mathbf{N} \le 300$

Sample

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

```
1
3
3 2 1
```

Problem E. Google Royale

Small input 20 points	Solve E-small
Large input 40 points	Solve E-large

Problem

While visiting the planet Theta VIII, your team of space explorers is forced to participate in the plot of a badly-written book, which takes place in a hotel/casino called the *Google Royale*. In order to escape the Royale, you will have to make enough money from gambling that you can buy the hotel for **V** dollars and leave.

You start with **A** dollars, and you will participate in *betting rounds* until one of two conditions is met. If you finish any betting round with ≤ 0 dollars, you will lose; if you finish a betting round with ≥ **V** dollars, you will buy the hotel and leave. Otherwise you'll keep starting new betting rounds.

Each betting round consists of one or more coin flips. If you have X dollars at the start of the round, you can choose any integer B between 1 and `min(X, M)` to bet on the first coin flip.

With probability 50%, you win the coin flip, and the Royale immediately pays you B dollars. You now have `X + B` dollars, and the betting round ends.

With probability 50%, you lose the coin flip and owe the Royale B dollars. You can now pay the B dollars you owe and end the round. Or if `2B ≤ M`, you can instead delay the payment and do a second coin flip with twice the bet: 2B dollars. If you lose again, then you owe the Royale B+2B=3B dollars. You can continue doubling your bet in this way to 4B, 8B, etc., until either you win a coin flip, you choose to stop, or your next bet would exceed **M**. You can even continue if the total of all your bets in the current betting round exceeds X.

Once the round is over, you must pay the Royale for each coin flip you lost, and if you won a coin flip, the Royale pays you for that. For example, if you start with a bet of 1 dollars, lose three coin flips, and then win one, you would gain $8 - $4 - $2 - $1 = $1. If you lose three coin flips and then stopped, you would lose $4 + $2 + $1 = $7. If you are left with $0 or less after paying, then you are broke, and you have just lost the game.

Luckily you have brought an android with you, and he is able to compute the probability that you will win if you follow an optimal strategy. What is that probability, and what is the largest possible first bet you could make to have that probability? Remember that you are not allowed to bet more than **M**!

Example

Suppose that you decide to use the following (sub-optimal) strategy. You have **A**=5 dollars; **M**=20 and V=40. The following sequence of events is possible:

- Round 1: You can start by betting 1, 2, 3, 4 or 5 dollars. You decide to begin a betting round by betting 2 dollars.
  - Step 1 (B=$2): You win the first coin flip. You gain 2 dollars, and the betting round ends. Now you have 7 dollars.
- Round 2: You begin a betting round by betting 5 dollars.
  - Step 1 (B=$5): You lose the first coin flip. Now you owe the Royale 5 dollars. Since `5*2 ≤ 20`, you may do another coin flip with a bet of 5*2=10 dollars. You choose not to. You lose 5 dollars, and the betting round ends. Now you have 2 dollars.
- Round 3: You begin a betting round by betting 2 dollars.
  - Step 1 (B=$2): You lose. Now you owe the Royale 2 dollars. You choose to flip another coin with a bet of 4 dollars.
  - Step 2 (B=$4): You lose. Now you owe the Royale a total of 6 dollars. That's more than you have, which is okay. You choose to flip another coin with a bet of 8 dollars.
  - Step 3 (B=$8): You win. You gain 8 dollars, pay the 2+4=6 dollars you owe, and the betting round ends. Now you have 4 dollars.
- Round 4: You begin a betting round by betting 2 dollars.
  - Step 1 (B=$2): You lose. Now you owe the Royale 2 dollars. You choose to flip another coin with a bet of 4 dollars.
  - Step 2 (B=$4): You lose. Now you owe the Royale a total of 6 dollars. You choose to flip another coin with a bet of 8 dollars.
  - Step 3 (B=$8): You lose. Now you owe the Royale a total of 14 dollars. You choose to flip another coin with a bet of 16 dollars.
  - Step 4 (B=$16): You lose. Now you owe the Royale a total of 30 dollars. Since 2*16>**M**, you cannot flip another coin and you must pay what you owe. Now you have -26 dollars; you have lost.

Input

The first line of the input gives the number of test cases, **T**. **T** lines follow. Each line contains three integers separated by single spaces: **A**, **M** and **V**, in that order.

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 probability of winning if you follow an optimal strategy; and **z** is the maximum first bet you can make without reducing your probability of winning. **y** must be correct to within an absolute or relative error of 10^{-6}.

Limits

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

Small dataset

$1 \leq \mathbf{M} \leq 20$.
$1 \leq \mathbf{A} < \mathbf{V} \leq 20$.

Large dataset

$1 \leq \mathbf{M} \leq 10^{16}$.
$1 \leq \mathbf{A} < \mathbf{V} \leq 10^{16}$.

Sample

Input	Output
4 1 1 3 3 6 12 4 20 15 13 6 20	Case #1: 0.333333333 1 Case #2: 0.500000000 3 Case #3: 0.755555555 3 Case #4: 0.730769231 6