Anti-Blot System

Jimmy is a hard-working pupil in his second year at primary school. Recently he decided to convert all his notes into an electronic version. Sadly, he found that his math notes were full of ink blots.

He scanned the notes and sent them through his own OCR package (yes, he coded it all by himself at the age of 8). The OCR package replaced all ink blots by the string "`machula`".

Problem specification

You are given Jimmy's notes, processed by the OCR. They contain simple math exercises, which were used to practice addition on positive integers. Your task is to recover the damaged part of the notes.

Input specification

The first line of the input file contains an integer T specifying the number of test cases. Each test case is preceded by a blank line.

Each test case consists of exactly one line.
The line represents an equation of the form "`number + number = number`", where each `number` is a positive integer. One part of the equation will be replaced by the string "`machula`". The string always covers a contiguous non-empty sequence of digits, possibly even an entire number. You may assume that for each equation in the input there will be exactly one way to fill in the missing digits.

Output specification

For each test case, the output shall contain one line of the form "`number + number = number`". The line must represent the equation from that test case with all missing digits filled in.

Example

input

```
3

23 + 47 = machula

3247 + 5machula2 = 3749

machula13 + 75425 = 77038
```

output

```
23 + 47 = 70
3247 + 502 = 3749
1613 + 75425 = 77038
```

Bus Story

"… and they lived happily ever after."
"But that could not ever happen", screamed the children lying in bed. "We want something that could really take place somewhere, not just a commercially successful reprint of fairy tales several hundreds of years old", said a nine year old boy.
Yeah, the times they are a-changing.

"So, let's play a game" said mother, trying to hide the fact that she has already missed 15 minutes of her favorite soap opera. This has to finish soon! "I will tell you a story and all you need to do is to tell me whether it can be true or not. If you tell me the correct answer, I will let you watch TV instead." Children quickly agreed and mother started telling a story:

"Once upon a time, there was a bus with some passengers. After it has stopped on a bus stop, a man got inside. On the following stop, 7 people got off and another 4 boarded the bus. …

… and on the terminal stop, two travellers got out, the driver locked the empty bus and went away too."

After the story was over, the children slowly began to realize what they have agreed upon. Telling whether the story might be true will not be an easy task.

Problem specification

You are given several bus travelling stories, similar to the one above. Each story will mention every change in the number of travellers during some period of time. Your task is to find out for each story whether it could really happen.

Input specification

The input file contains several stories. The stories are separated by a line consisting of the string "`NEW STORY`". This string does not occur inside any story.

The driver is not a passenger. He is in the bus during the entire story. Every passenger that boards the bus will travel for at least one stop. Each sentence ends with a period. Events mentioned in different sentences correspond to different bus stops.

Output specification

For each story output a single line containing the string "`YES`" if the story could be true and "`NO`" otherwise.

Example

input

```
The bus was empty, no people were sitting
inside except for the driver. On the next
stop, 2 people boarded and at the same
time, 4 of them left bus. The next stop
was terminal, after arrival the driver
locked the bus and went away.
NEW STORY
On this stop, 2 people boarded and while 4
left the bus. At the terminal, 3
people left, the driver locked the bus and
went away.
```

output

```
NO
YES
```

In the first case, after the first stop the number of passengers would be -2, and this is clearly impossible. In the second case, the story was possible if the bus contained 5 people at the beginning of the story.

Calling Me Calling You

Banana Immobile[2] the newest star on the cellular sky, is starting to provide a new service. First, they collected statistics of the calls made by their customers during a few months and noted which pairs communicated very often. Then, they started offering the Happy Packs.

When a customer buys a Happy Pack, he is presented with a list of phone numbers he called or was called from during the last month. He may now mark at most K of these phone numbers as his "Happy numbers". Making a call to a Happy number will then cost him absolutely nothing! One person can even buy more than one Happy Pack and mark up to K new phone numbers for each Happy Pack bought. Wonderful, isn't it?

A group of students who would spend hours on the phone (if they had the money to pay the bills) learned about this new service from the local newspaper. They started spinning a plan immediately: Suppose that you could choose two Happy numbers for each Happy Pack and that Henry and Alice bought one each. Henry would then specify Oscar's and Chad's numbers as his Happy ones, while Alice would choose Patricia's and Chad's. Then Patricia could call Oscar free of charge!

How's that possible? All their phones are able to arrange conference calls. In other words, each phone is able to take two phone calls it participates in, and merge them into one larger call where every participant of both original calls can hear all the other participants.

In our example Patricia could start by ringing Alice. Alice does not pick up the call (as this would cost Patricia some money). Instead, Alice calls her back, and Patricia explains that she needs to talk to Oscar. Alice calls Chad and merges the two calls. So now Patricia, Alice and Chad are having a conference call. Chad rings Henry, Henry calls Chad back, Chad merges the two calls he takes part in. Henry calls Oscar and merges the two calls together. Now all five friends share the same call and Patricia can (finally!) talk to Oscar for free.

Simple, isn't it? Unfortunately, these five friends belong to a much bigger group. Figuring out who needs to buy the Happy Pack (and how many of them) and choosing the Happy numbers is a very difficult task, so they'd like to ask you for help.

Problem specification

Given is a group of friends. For each of them, you know the list of numbers that are eligible to be his Happy numbers. Find the least number of Happy Packs they need to buy so that all of them can call each other for free. You also have to provide one optimal way of achieving their goal. You can assume that the problem has at least one solution.

Input specification

The first line of the input file contains an integer T specifying the number of test cases. Each test case is preceded by a blank line.

The first line of each test case contains three integers, N and M, and K, where N is the number of friends, M is explained below, and K specifies how many Happy numbers you get in one Happy Pack.

For simplicity we will assume that the friends are numbered from 1 to N. The next M lines of the test case contain two integers A and B each. These numbers denote that one of the friends A and B called the other at least once during the last month. This means that A can pick B's number and also that B can pick A's number as her Happy number.

Output specification

For each of the test cases, the output shall consist of multiple lines. Each line with the exception of the last one should contain two integers, A and B specifying that person A should choose person B's phone number as his or her Happy number. The last line should contain two zeroes.

Note that the number of Happy Packs necessary can be easily computed from your output data. Your only goal is to minimize the number of the Happy Packs. If there are more ways to achieve this goal you may print an arbitrary one. The pairs in the output don't have to be sorted in any way, and their count does not have to be minimal (as shown in the second test case in the example). Although it doesn't make much sense, you can include the same pair multiple times in the output. Be aware that each such occurence will be counted separately (and might increase the number of Happy Packs undesirably).

Example

input

```
2

5 4 2
1 2
4 5
3 2
3 4

5 6 3
1 2
1 3
1 4
4 5
2 3
3 5
```

output

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

Delicious Cake

Lenka likes to bake cakes since her childhood, when she has learned to bake from her mom. She soon became a cake expert able to bake chocolate cakes, apple pies, muffins, cookies, cheese cakes, tortes and many other cakes.

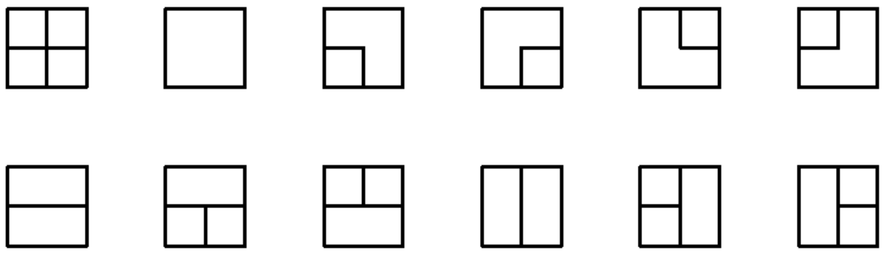
Recently, she has started her studies of math at Comenius University in Bratislava. In the first year she is taking combinatorics class. Today she is studying for the final exam. Since the brain needs a lot of sugar to study math, she has baked, just for herself, her favorite, very delicious, strawberry cake.

The cake, still hot, is lying on an $N \times M$ inch sheet pan. Hungrily waiting for the cake to cool off Lenka came up with an interesting combinatorial question: How many different possibilities to cut the cake are there so that every connected piece consists of some number of 1×1 inch unit squares?

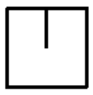
Problem specification

The cake can be viewed as a grid consisting of $N \times M$ unit squares. We are allowed to cut the cake along the grid lines. As a result the cake splits into several connected pieces. (Two unit squares remain connected if they share a side which was not cut.) How many different ways are there to cut the cake? We consider two cuttings of the cake to be the same if the resulting connected pieces of both cuttings have the same shape and are at the same positions within the cake. In other words, we are only counting those cuttings where no cut leads between two unit squares that are in the same connected piece.

The following picture ilustrates all the 12 different possible ways how to cut a 2×2 inch cake:



Note that cutting, for example, as on following picture is the same as not cutting at all.



Input specification

The first line of the input file contains an integer T specifying the number of test cases. Each test case is preceded by a blank line.

Each test case consists of a single line with two positive integers N and M – dimensions of the cake.

Output specification

For each test case output a line with a single positive integer – the number of different possibilities how to cut the cake.

Example

input

```
2

1 2

2 2
```

output

```
2
12
```

Enhancing IPSC Rules

We would like to change the scoring rules for IPSC in the future. The goal is that no two teams would end on the same position. To achieve this we would like the tied teams to play a game of Rock, Paper, Scissors (see `http://en.wikipedia.org/wiki/Rock_paper_scissors`). We would like the teams to submit their choice as a simple picture, but we don't have the means to check who has won.

Problem specification

You are given several images. Each image shows a single game of Rock, Paper, Scissors, which consists of several rounds. Your task is to determine the result of each game.

Input specification

The input file is an archive containing several images named `e1_XX.png` for the easy input and `e2_XX.png` for the hard input. Each image contains a single game, and the `XX` in its filename is the number of this game. The games are numbered from zero.

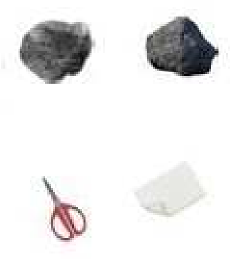
Output specification

Process the games ordered by their number. For each game output a line containing the word `LEFT` if the left-hand player has won, `RIGHT` if the right-hand player has won, or `DRAW` in case of a draw.

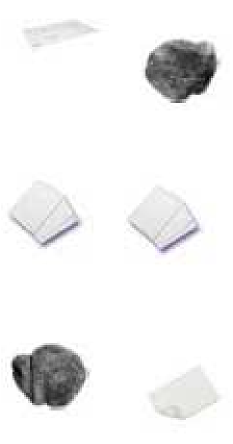
Example

input

`e0_00.png`



`e0_01.png`



output

```
LEFT
DRAW
```

In the first game the left player won 1:0, in the second game the final score was 1:1.

Finally some Sudoku

In case you are not familiar with the Sudoku puzzle, please refer to the last section of this problem statement before reading any further.

We would like to present our variation: the Cross-linked Sudoku.

In Cross-linked Sudoku you are given several Sudoku puzzles at once. In addition, some of the squares in some of the puzzles are colored. Your task is to solve the entire set of puzzles with the following additional restriction: *Squares sharing the same color must contain the same number.*

The restriction remains in place even if the squares are on different boards. The colors are independent, squares with different colors may contain the same number.

Problem specification

Given a Cross-linked Sudoku puzzle, find its solution. You may assume that the solution is unique.

Input specification

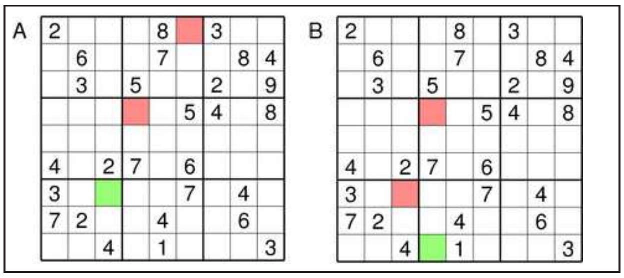
The input file is an image file containing the puzzle. Note that the Sudoku boards in the input file are labeled by capital letters.

Output specification

After solving the puzzle, order the boards by their letters. For each board, write the 9×9 numbers it contains in row major order. Separate the numbers by some whitespaces.

Example

input



output

```
2 4 5 9 8 1 3 7 6
1 6 9 2 7 3 5 8 4
8 3 7 5 6 4 2 1 9
9 7 6 1 2 5 4 3 8
5 1 3 4 9 8 6 2 7
4 8 2 7 3 6 9 5 1
3 9 8 6 5 7 1 4 2
7 2 1 3 4 9 8 6 5
6 5 4 8 1 2 7 9 3

2 4 5 9 8 1 3 7 6
1 6 9 2 7 3 5 8 4
8 3 7 5 6 4 2 1 9
9 7 6 1 2 5 4 3 8
5 1 3 4 9 8 6 2 7
4 8 2 7 3 6 9 5 1
3 9 1 6 5 7 8 4 2
7 2 8 3 4 9 1 6 5
6 5 4 8 1 2 7 9 3
```

The solution for the example above is unique. Note that without the colored squares each of the separate puzzles would have more than one possible solution.

Sudoku rules

In a classic Sudoku puzzle, you are given a 9×9 grid that is partially filled with numbers 1 to 9:

2				8		3		
	6			7			8	4
	3		5			2		9
			1		5	4		8
4		2	7		6			
3		1			7		4	
7	2			4			6	
		4		1				3

Your task is to fill in the remaining squares in such a way that in the end:

- each square contains exactly one number (between 1 and 9, inclusive),
- each row contains each of the numbers exactly once,
- each column contains each of the numbers exactly once, and
- each of the nine highlighted 3×3 squares contains each of the numbers exactly once.

For the puzzle above, the only valid solution is:

2	4	5	9	8	1	3	7	6
1	6	9	2	7	3	5	8	4
8	3	7	5	6	4	2	1	9
9	7	6	1	2	5	4	3	8
5	1	3	4	9	8	6	2	7
4	8	2	7	3	6	9	5	1
3	9	1	6	5	7	8	4	2
7	2	8	3	4	9	1	6	5
6	5	4	8	1	2	7	9	3

Guess the Numbers

Ever since Jan learned his numbers, he loved to make number riddles. Already as a small boy, he annoyed everyone with them. Right now, he made up a new guessing game and came to annoy you. You have to guess the numbers that Jan thinks.

Problem specification

Jan thinks of a sequence of N non-negative integers. Each of these integers has at most D digits. Your task is to guess the entire sequence using as few submits as possible. You can make two types of submits:

1. *Question.* You submit a sequence of N non-negative integers. Each of these integers can have at most G digits. As the evaluation result you will receive the scalar product of Jan's sequence and the sequence you submitted. (See below for a clarification if you don't know what a scalar product is.)
2. *Guess.* You submit a sequence of N non-negative integers. Each of these integers can have at most D digits. As the evaluation result you will be notified whether you guessed the entire sequence correctly.

The scalar product of two sequences $(X_1, \dots, X_N)$ and $(Y_1, \dots, Y_N)$ can be calculated as follows:

$$(X_1, \dots, X_N) \cdot (Y_1, \dots, Y_N) \; = \; X_1Y_1 + \dots + X_NY_N$$

This means that Jan takes his first number and multiplies it by your first number, then he takes his second thought number and multiplies it by your second number and so on. In this way he will get N numbers. The scalar product of the two sequences is the sum of these N numbers.

Input specification

The input file contains three positive integers: N, D and G separated by a single space. N is the length of the sequence that Jan is thinking of. D is the maximal number of digits of each of Jan's numbers. G is the maximal number of digits in the numbers you may use in your *questions*.

Output specification

The output file you will be submitting must contain exactly $N+1$ rows. The first row must contain a zero (0) if the submit is a *question*, or a one (1) if the submit is a *guess*. The next N rows must contain the sequence you want to submit. More precisely, each of the next N rows must contain one non-negative integer. The integer must not contain unnecessary leading zeroes. If the submit is a *question*, each of the integers must contain at most G digits. If it is a *guess*, each of the integers must contain at most D digits.

Evaluation result

For a malformed submit the evaluation result will be an appropriate error message.

For a *question* the answer will be one non-negative integer – the scalar product of Jan's sequence and the sequence you submitted. Note that the answer will always be exact and that it may contain more than G digits.

For an incorrect *guess* the evaluation result will be Wrong answer. For a correct *guess* the evaluation result will be OK, and you will receive points for solving the corresponding input.

Note that for each input you may only make **ten (10) submits**, i.e., at most 9 questions and a guess.

Example

input

```
3 4 5
```

Note that for readability each of your submits was formatted to fit on a single line.

submits

```
You: 0 7 3 5
Jan: 28
You: 0 6 4 1
Jan: 17
You: 0 0 2 2
Jan: 10
You: 1 1 2 3
Jan: OK
```

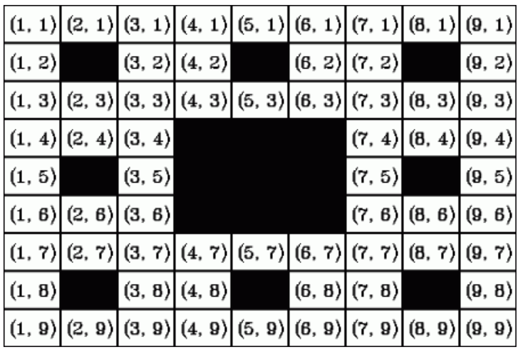
Here-There

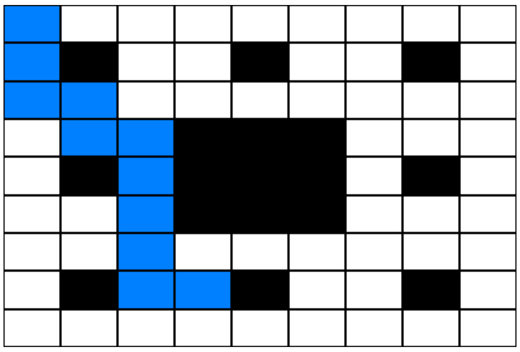
Do you know the game Here-There? I presume you don't. It's a virtual board game, so you should first learn how this virtual board looks like.

The process of making the board is remarkably simple. You start by taking a square with side of length 3^N, divide it into nine smaller squares of equal size and remove the central one. Then, you repeat the same divide-and-remove-the-centre process with each of the eight smaller squares over and over (N times in total), until you are left with a grid that consists of many little squares with side length 1 – and of many holes. By the way, the number N is called the degree of the board.

The game itself consists of two steps. First, your opponent chooses two squares on the board, one of them will be "Here" and the other one "There". Your task is to estimate the least number of steps you have to take if you started Here and wanted to get to There. One step consists of moving to another square, which has a common side with the one you're standing on. Obviously, you cannot move over the removed parts of the board. If you guess the number of steps correctly, you get a point.

You would really like to become a master of this game, so you have written down the sizes of the boards and the positions of the Here and There squares from several games in the past. Now, you'd like to find the exact number of steps you need to take to get from Here to There on each of the boards. Each square is described by two numbers between 1 and 3^N, the first of them denoting the column and the second one the row the square is in. The square in the upper left corner of the board has coordinates $(1, 1)$, as you can see on the picture below.





You can see one of the shortest paths between squares $(1, 1)$ and $(4, 8)$ on the picture, consisting of 10 steps.

Problem specification

You are given several boards and pairs of squares on them and your task is to find the steps-distances between the squares in each pair.

Input specification

The first line of the input file contains an integer T specifying the number of test cases. Each test case is preceded by a blank line.

Each test case consists of five integers D, H_c, H_r, T_c, T_r, specifying the degree of the board D, the coordinates of Here (H_c, H_r) and the coordinates of There (T_c, T_r).

Output specification

For each of the test cases, output a single line with one integer – the steps-distance between Here and There.

Example

input

```
2

1 1 2 2 1

2 1 1 4 8
```

output

```
2
10
```

Internet is Faulty

David has a problem. He wants to transfer a big file through the internet from his home computer to his computer at work. The size of his file is S kilobytes.

The internet consists of N computers, numbered from 1 to N. David's home computer has the number 1 and his computer at work has the number 2. Some pairs of computers are connected by different types of links (such as network cables or wi-fi). Some of these links (e.g., a satellite dish) may be unidirectional, thus for simplicity we will assume that all links are unidirectional.

The data are sent across the network in packets, each packet contains exactly one kilobyte of data. For each link David knows how faulty it is, i.e. the probability that a network packet gets from one computer to the other through the link. We assume that all transfers are independent from each other. That is, regardless of whether the previous packet was transferred successfully or not, the probability that the next one will pass through remains the same.

Since the network is faulty and the work computer might be many links away from the home computer, the transfer of David's file along even the best route between the two computers might take too long. Luckily, David has an account on some of the machines in the network. He may use these machines as temporary storage, and thus shorten the time of the transfer.

The file transfer will consist of several steps. In each step, David selects a series of links starting with a computer that already has the file and ending with a computer David has an account on. Prior to the transfer, the file is split into S packets. Then the packets are sent one after another along the chosen route. The probability that a packet successfully arrives at the destination computer is the product of probabilities that it passes all the links. If the packet is lost it is resent immediately. Each attempt to send a packet, successful or unsuccessful, takes exactly one millisecond, regardless of the number of links on the route. After the entire file is transferred, David may start another transfer from the new machine, and so on.

Problem specification

You are given the number of computers N and the file size S. For each pair of computers u, v we know the probability $p(u, v)$ that a network packet passes successfully through the direct link from computer u to v. (The value zero means that there is no direct link from u to v.) Finally, you are given a list of servers where David has an account.

Find a way how to send David's file so that the expected transfer time is minimized, and output this expected time in milliseconds. You may assume that the expected transfer time is less than 1 000 000 000 (1 billion) milliseconds.

Input specification

The first line of the input file contains an integer T specifying the number of test cases. Each test case is preceded by a blank line.

Each test case looks as follows: The first line contains a positive integer $N \geq 2$ giving the number of computers on the internet.

N lines follow. The i-th of these lines contains N integers, each of them between 0 and 100, inclusive. These integers give the probabilities $p(i, 1), p(i, 2), \ldots, p(i, N)$ in percents.

The next line contains a non-negative integer M – the number of computers on which David has an account. The following line contains M integers – the numbers of these computers. This list will always contain the integers 1 and 2 (corresponding to David's home and work computer).

The last line contains an integer S – the size of David's file in kilobytes.

Output specification

For each test case output a single line with a real number – the expected time of the transfer in milliseconds when using the best possible strategy.

Each number in the output file should have sufficiently many decimal places. We recommend printing all results rounded to seven decimal places. Your output will be considered correct if each number has an absolute or relative error less than 10^{-6}.

Example

input

```
2

4
0 0 40 66
0 0 0 30
40 47 0 66
0 30 66 0
4
1 2 3 4
47

5
0 1 20 0 0
0 0 0 0 0
0 0 0 50 90
0 20 0 0 0
0 0 0 90 0
3
1 2 5
10
```

output

```
207.897153
111.111111
```

In the second case, we have four possible strategies.

The first one is to use the direct link from 1 to 2. This link is really really faulty, and the expected time for this solution is 1000 milliseconds. The second strategy is to transfer the file from computer 1 to computer 2 using the route 1-3-4-1. The probability that a packet passes this route is $20\% \cdot 50\% \cdot 20\% = 2\%$, and thus the expected transfer time is 500 milliseconds. The third strategy is to use the route 1-3-5-4-2. Here the probability of successfully transfering a packet is 3.24% and the expected transfer time is roughly 308.6 milliseconds.

The optimal solution is to use the computer 5 as temporary storage. We will first transfer the file along the route 1-3-5, and then along the route 5-4-2. For each of the transfers the probability is $20\% \cdot 90\% = 18\%$, and thus the expected time of each of the transfers is 55.5555 milliseconds.

jPix

jPix is a new, very promising, open source bitmap editor. As the project is in a very alpha stage of development, currently jPix supports only 3 colors: white, grey and black. Other features of jPix include drawing straight lines with grey color and floodfill with black color. However, the most promising and coolest feature of jPix is the ability to work with images of vast sizes.

To draw lines jPix uses the following advanced algorithm:

```
void draw_line(int x1, int y1, int x2, int y2)
{
        int x_middle = (x1+x2)/2;
        int y_middle = (y1+y2)/2;

        set_color(GREY);
        put_pixel(x_middle, y_middle);

        if((x1!=x2) || (y1!=y2))
        {
                if(abs(x1-x2) > abs(y1-y2))
                {
                        if(x1<x2)
                        {
                                draw_line(x1, y1, x_middle, y_middle);
                                draw_line(x_middle+1, y_middle, x2, y2);
                        }
                        else // x1>x2
                        {
                                draw_line(x1, y1, x_middle+1, y_middle);
                                draw_line(x_middle, y_middle, x2, y2);
                        }
                }
                else
                {
                        if(y1<y2)
                        {
                                draw_line(x1, y1, x_middle, y_middle);
                                draw_line(x_middle, y_middle+1, x2, y2);
                        }
                        else // y1>y2
                        {
                                draw_line(x1, y1, x_middle, y_middle+1);
                                draw_line(x_middle, y_middle, x2, y2);
                        }
                }
        }
}
```

One of the features that is urgently needed among its users is the ability to report the number of pixels of each color in the image. The developers of jPix, unable to come up with an efficient solution, asked us, the IPSC team, for help. Since we are lost too, we ask you to help us.

Problem specification

A jPix image is X pixels wide and Y pixels high. The pixel $(0, 0)$ is the upper left corner, and the pixel $(X-1, Y-1)$ is the lower right corner. When jPix starts, the image is completely white.

The user is allowed to draw several line segments using grey color. The segments, however, must form a simple polygon. (See the input specification for a more precise definition.) Then the user must issue the flood fill command, starting from pixel $(0, 0)$; the fill color is black. (Since jPix is quite buggy, any other sequence of commands crashes jPix.)

Given the list of the commands issued by the user, your task is to implement the requested feature which will count the number of pixels of each color.

Input specification

The first line of the input file contains an integer T specifying the number of test cases. Each test case is preceded by a blank line.

Each test case looks as follows: The first line contains two positive integers X and Y, the width and height of the image. The following line contains N – the number of line segments drawn by the user. Each of the following N lines contains four positive integers x_1, y_1, x_2, y_2 describing the two endpoints (x_1, y_1) and (x_2, y_2) of a line segment. You may assume that the line segments are given in order and correctly describe a simple polygon which is not touching the border of the image.

More precisely, you may assume the following things:

- The exact mathematical object defined by the N line segments is a simple polygon, that is, any pair of consecutive edges has one point in common (the common vertex) and no pair of edges that are not consecutive has a point in common.
- The lines are given in the order in which they occur on the polygon's boundary. That is, the second line must start where the first ended, third must start where the second ended etc., and the last line must end where the first one started.
- After the lines are painted in jPix, no grey pixel will lie on the border of the image.

Output specification

For each test case output a line with three positive integers – the number of pixels of white, grey, and black color, respectively.

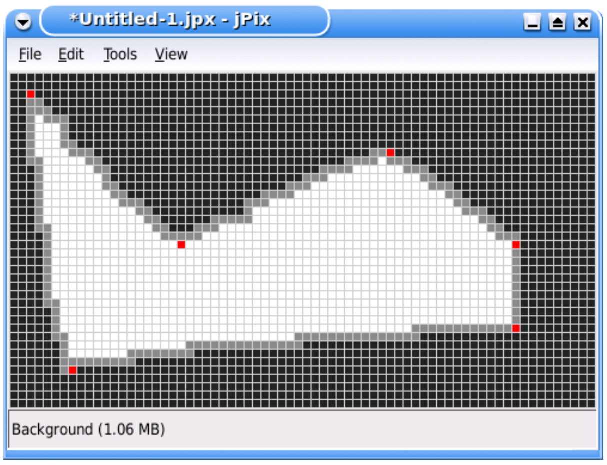
Example

input

```
3

10 10
3
3 3 8 8
8 8 8 3
8 3 3 3

10 10
4
2 2 6 2
6 2 6 6
6 6 2 6
2 6 2 2

70 40
6
2  2  20 20
20 20 45 9
45 9  60 20
60 20 60 30
60 30 7  35
7  35 2  2
```

output

```
7 20 73
9 16 75
931 202 1667
```



The image shows a typical jPix session corresponding to the third example input. For your convenience the line endpoints are shown in red. In the actual image produced by jPix these points are grey.

Know Your Crypto

Note to self: never make problem statements at the last possible moment.

In this problem your task is to examine the given input file, make some sense out of it and recover the correct answer. For each input file the answer is a single English word.

We will not tell you anything about the easy input, it is already easy enough.

And we have to apologize for the hard problem, it is probably impossible to solve. This task was prepared at the last possible moment, and we didn't have any idea what to do with the hard input. So finally we decided that we will just randomly change all the letters of the plain text. This is the program we used:

```
#include <stdio.h>
#include <ctype.h>
#include <stdlib.h>
#include <time.h>
char ch;

char change(char ch) { return 'a'+((ch-'a'+(rand()%26))%26); }

int main() {
  srand(time(NULL));
  while (scanf("%c",&ch)==1) if (isalpha(ch)) printf("%c",change(tolower(ch)));
  printf("\n");
  return 0;
}
```

Of course, this is not completely random, the `rand()` function is only returning pseudo-random values. If you think it might help you, our machine used the `rand()` implementation from the GNU C library, version 2.4 (`http://ftp.gnu.org/gnu/glibc/glibc-2.4.tar.gz`). We even wrote down some additional notes on how this pseudo-random function works (see the next page). Feel free to spend your valuable contest time reading them :)

Output specification

After finding the solution for an input (a single English word), create an output file with a single line containing this word **in lowercase**.

Example

input	output
`the answer is a male cow`	`bull`

Some random musings on rand()

Do all C compilers have the same `rand()` function?
No. The standard does not define a fixed implementation.

How can I tell two machines have the same `rand()` function?
The simplest test would be to print out some random values for a fixed seed. If they match, you most probably have the same implementation. E.g., you may try this piece of code:

```
#include <stdio.h>
#include <stdlib.h>

int main() {
  int i;
  srand(47);
  for (i=0; i<10; i++) printf("%d\n",rand());
  return 0;
}
```

What does that code output on your machine?
The output is the following sequence:

```
1773134790
1924001091
438623483
50210213
809830442
761766184
2023073002
729207936
708098155
2037448065
```

I don't have a C compiler that uses the GNU C library.
Can I implement the same pseudo-random generator on my machine?

The GNU C library is distributed under the GPL. You can just download it at `http://ftp.gnu.org/gnu/glibc/glibc-2.4.tar.gz` and look at the implementation. (It's in the directory `stdlib`, look for filenames containing the substring `rand`.)

Alternately, we even provide our own reference implementation below. The `ipsc_srandom` and `ipsc_random` functions are equivalent to the GNU C library functions `srand` and `rand` we used.

```
#define GENERATOR_DEG 31
int32_t ipsc_random_table[GENERATOR_DEG] = {
 -1726662223,   379960547,  1735697613,  1040273694,  1313901226,  1627687941,
  -179304937, -2073333483,  1780058412, -1989503057,  -615974602,   344556628,
   939512070, -1249116260,  1507946756,  -812545463,   154635395,  1388815473,
 -1926676823,   525320961, -1009028674,   968117788,  -123449607,  1284210865,
   435012392, -2017506339,  -911064859,  -370259173,  1132637927,  1398500161,
  -205601318,
};
int front_pointer=3, rear_pointer=0;

int32_t ipsc_random() {
  int32_t result;
  ipsc_random_table[ front_pointer ] += ipsc_random_table[ rear_pointer ];
  result = ( ipsc_random_table[ front_pointer ] >> 1 ) & 0x7fffffff;
  front_pointer++, rear_pointer++;
  if (front_pointer >= GENERATOR_DEG) front_pointer = 0;
  if (rear_pointer >= GENERATOR_DEG) rear_pointer = 0;
  return result;
}

void ipsc_srandom(unsigned int seed) {
  int32_t i, dst=0, kc=GENERATOR_DEG, word, hi, lo;
  word = ipsc_random_table[0] = (seed==0) ? 1 : seed;
  for (i = 1; i < kc; ++i) {
    hi = word / 127773, lo = word % 127773;
    word = 16807 * lo - 2836 * hi;
    if (word < 0) word += 2147483647;
    ipsc_random_table[++dst] = word;
  }
  front_pointer=3, rear_pointer=0;
  kc *= 10;
  while (--kc >= 0) ipsc_random();
}
```

Little Peter's Tower

My little brother Peter likes playing with his building set. He has a huge bag containing cylinder-shaped building blocks of various dimensions. He wants to build a very tall tower out of them. But that is not as simple as it looks.

For each two integers $0 < r \leq R$ and $0 < h \leq H$ there are infinitely many cylinder-shaped building blocks with base radius r and height h in his bag. He is building the tower by following the following rule:

Each minute he draws one block from his bag. Then he has to decide among two options: either he places this block on top of his current tower (or on the floor if this is the first block to be used), or he returns the block into his bag.

He may only place the block on top of the current tower if its base radius is less than or equal to the radius of the top of the the tower. If the block he picked out of the bag is too large to fit on top of the tower, he has to return it to the bag.

Peter can not predict or influence the dimensions of the block he will draw from the bag. For each block type (radius r and height h) the probability that he will draw it in the next minute is equal to $1/(R * H)$.

Peter's goal is to build a tower that's as tall as possible. However, he needs to eat and sleep (or at least his mother claims so). Thus his playing time is limited, and Peter knows the exact number of minutes he has left.

Peter is already an experienced tower builder, and he decided to use the building strategy that will maximize the average height of the complete tower.

Problem specification

Given the maximum dimensions of a building block and the time Peter has to build his tower, compute the maximum average height of the complete tower. (The maximum is taken over all decision strategies and average is taken over the drawn block dimensions.)

Input specification

The first line of the input file contains an integer T specifying the number of test cases. Each test case is preceded by a blank line.

Each test case consists of a single line with three positive integers: R (maximum base radius of a building block), H (maximum height of a building block) and T (time in minutes he has to build his tower).

Output specification

For each test case output a single line with a single (real) number – the answer for that test case.

Each number in the output file should have sufficiently many decimal places. We recommend printing all results rounded to seven decimal places. Your output will be considered correct if each number has an absolute or relative error less than 10^{-6}.

Example

input

```
2

1 10 10

3 7 4
```

output

```
55.00
11.25937
```