# Balanced Number

Problem Description

A balanced number is a non-negative integer that can be balanced if a pivot is placed at some digit. More specifically, imagine each digit as a box with weight indicated by the digit. When a pivot is placed at some digit of the number, the distance from a digit to the pivot is the offset between it and the pivot. Then the torques of left part and right part can be calculated. It is balanced if they are the same. A balanced number must be balanced with the pivot at some of its digits. For example, 4139 is a balanced number with pivot fixed at 3. The torqueses are 4*2 + 1*1 = 9 and 9*1 = 9, for left part and right part, respectively. It's your job
to calculate the number of balanced numbers in a given range [x, y].

Input
The input contains multiple test cases. The first line is the total number of cases T ($0 < T \leq 30$). For each case, there are two integers separated by a space in a line, x and y. ($0 \leq x \leq y \leq 10^{18}$).

Output
For each case, print the number of balanced numbers in the range [x, y] in a line.

Sample Input

```
2
0 9
7604 24324
```

Sample Output

```
10
897
```

# Battle over Cities

Problem Description
It is vitally important to have all the cities connected by highways in a war, but some of them are destroyed now because of the war. Furthermore,if a city is conquered, all the highways from/toward that city will be closed by the enemy, and we must repair some destroyed highways to keep other cities connected, with the minimum cost if possible.
Given the map of cities which have all the destroyed and remaining highways marked, you are supposed to tell the cost to connect other cities if each city is conquered by the enemy.

Input
The input contains multiple test cases. The first line is the total number of cases T (T ≤ 10). Each case starts with a line containing 2 numbers N (0 < N ≤ 20000), and M (0 ≤ M ≤ 100000), which are the total number of cities, and the number of highways, respectively. Then M lines follow, each describes a highway by 4 integers: City1 City2 Cost Status where City1 and City2 are the numbers of the cities the highway connects (the cities are numbered from 1 to N), Cost (0 < Cost ≤ 20000) is the effort taken to repair that highway if necessary, and Status is either 0, meaning that highway is destroyed, or 1, meaning that highway is in use.
Note: It is guaranteed that the whole country was connected before the war and there is no duplicated high ways between any two cities.

Output
For each test case, output N lines of integers. The integer in the i-th line indicates the cost to keep the cities connected if the i-th city is conquered by the enemy. In case the cities cannot be connected after the i-th city is conquered by the enemy, output "inf" instead in the corresponding place.

Sample Input
```
3
4 5
1 2 1 1
1 3 1 1
2 3 1 0
2 4 1 1
3 4 2 0
4 5
1 2 1 1
1 3 1 1
2 3 1 0
2 4 1 1
3 4 1 0
3 2
1 2 1 1
1 3 1 1
```

Sample Output

```
1
2
0
0
1
1
0
0
inf
0
0
```

# Binary Number

Problem Description
For 2 non-negative integers x and y, f(x, y) is defined as the number of different bits in the binary format of x and y. For example, f(2, 3)=1,f(0, 3)=2, f(5, 10)=4. Now given 2 sets of non-negative integers A and B, for each integer b in B, you should find an integer a in A such that f(a, b) is minimized. If there are more than one such integer in set A, choose the smallest one.

Input
The first line of the input is an integer T ($0 < T \leq 100$), indicating the number of test cases. The first line of each test case contains 2 positive integers m and n ($0 < m, n \leq 100$), indicating the numbers of integers of the 2 sets A and B, respectively. Then follow ($m + n$) lines, each of which contains a non-negative integers no larger than 1000000. The first m lines are the integers in set A and the other n lines are the integers in set B.

Output
For each test case you should output n lines, each of which contains the result for each query in a single line.

Sample Input

```
2
2 5
1
2
```

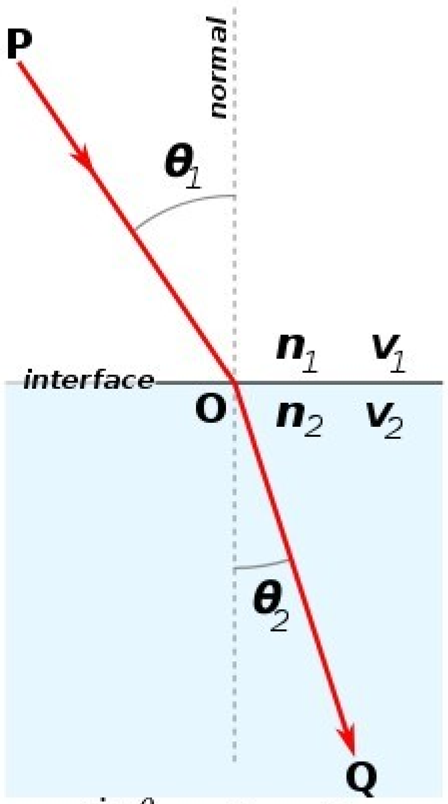
```
1
2
3
4
5
5 2
1000000
9999
1423
3421
0
13245
353
```

## Sample Output

```
1
2
1
1
1
9999
0
```

# Detector Placement

Problem Description
Dr. Gale is testing his laser system. He uses a detector to collect the signals from a fixed laser generator. He also puts a special prism in the system so that he can filter the noise. But he is not sure where to put the detector to collect the signals. Can you help him with this problem?



$$\frac{\sin\theta_1}{\sin\theta_2} = \frac{v_1}{v_2} = \frac{n_2}{n_1}$$

Here n1 and n2 are the refractive indices of the two media. In order to simplify the problem, here we assume the prism is a triangle. The laser generator will not be placed on the surface of the prism or inside the prism. The laser goes in one direction and the detector can receive signals from any direction. The detector is place on the ground where th y-coordinate is zero. There is no energy lost in the refraction. That is to say, there is no reflection in the signal transmission. You can assume that there is no total reflection or the situation that the laser passes the vertex of the prism. Given the position and the direction of the laser generator and the prism, you are asked to find the position of detector so that it can receive the signals from the laser generator.

## Input

The input contains multiple test cases. The first line is the total number of cases T (T ≤ 100). The first line of each test case contains 2 integers, indicating the x and y coordinates of the laser generator respectively.The second line contains 2 integers describing a point the laser will go through when the prism is not placed. The third line contains 6 integers describing the three vertices of the prism. The fourth line contains a real number u, the refractive index of the prism (1 < u ≤ 10). We assume the refractive index of the air is always 1.0. The absolute value of the coordinates will not exceed 1000. The y coordinates are all nonnegative. The prism and the laser generator are strictly above the ground.

## Output

If there is no place in the ground that can receive the signals output "Error". Otherwise, output the x coordinate of the place. Your answer should be rounded to the 3rd digit after the decimal point.

## Sample Input

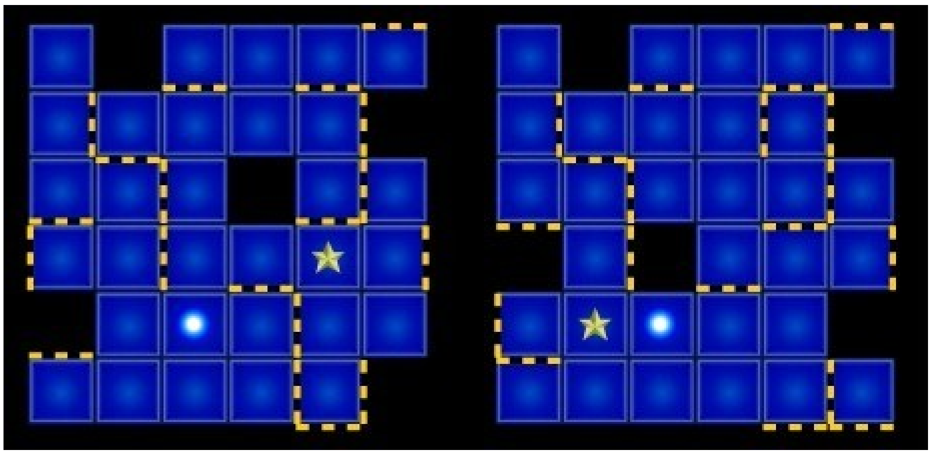
```
2
0 10
0 0
-1 3 1 2 -1 1
1.325
0 10
0 20
-1 3 1 2 -1 1
1.325
```

## Sample Output

```
-0.658
Error
```

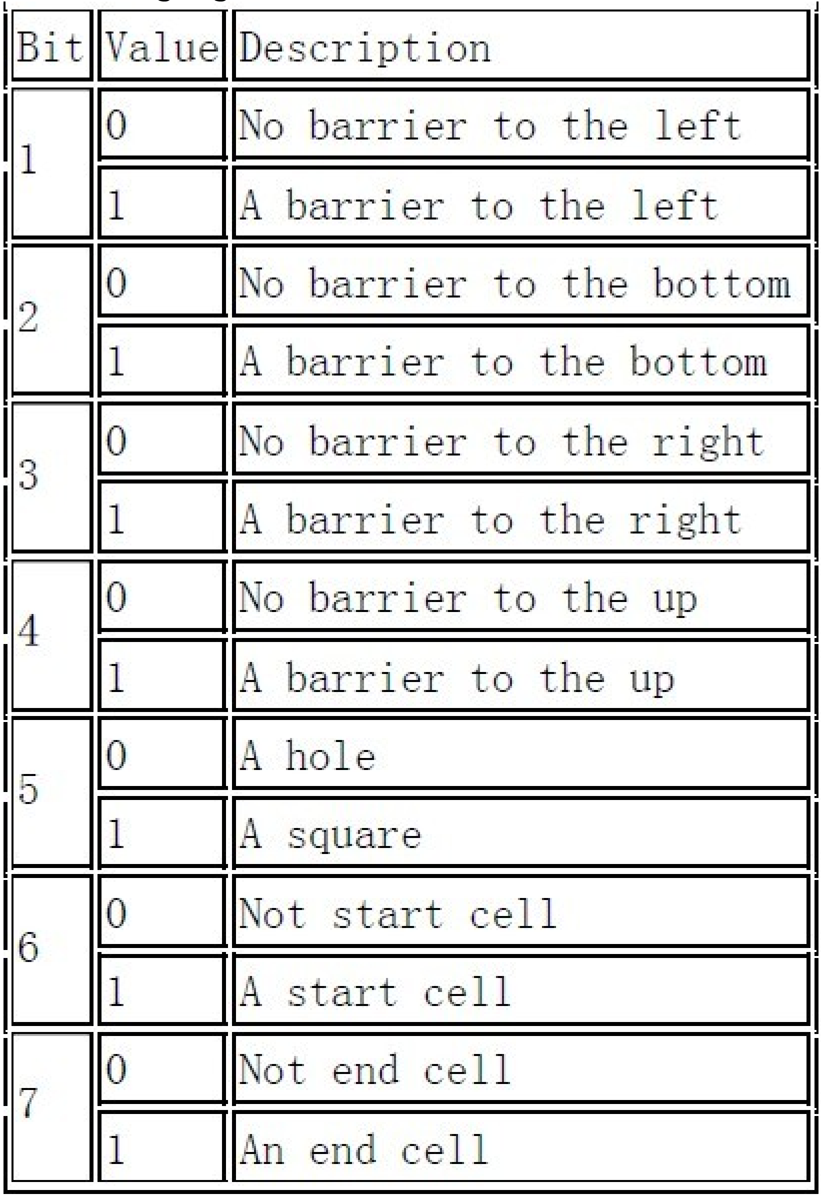
# Double Maze

Problem Description
Unlike single maze, double maze requires a common sequence of commands to solve both mazes. See the figure below for a quick understanding.



A maze is made up of 6*6 cells. A cell can be either a hole or a square. Moreover, a cell may be surrounded by barriers. There is ONLY one start cell (with a ball) and ONLY one end cell (with a star) in a single maze.These two cells are both squares. It is possible that the start cell and the end cell are the same one. The goal of a single maze is to move the ball from the start cell to the end cell. There are four commands in total,'L', 'D', 'R' and 'U' corresponding to moving the ball left, down, right and up one cell, respectively. The barriers may make the commands take no effect, i.e., the ball does NOT move if there is a barrier on the way.
When the ball gets to a hole or outside of the maze, it fails. A double maze is made up of two single mazes. The commands control two balls simultaneously, and the movements of two balls are according to the rules described above independently. Both balls will continue to move simultaneously if at least one of the balls has not got to the end cell.
So, a ball may move out of the end cell since the other ball has not been to the target. A double maze passes when both balls get to their end cells, or fails if

either of the two mazes fails. The goal of double maze is to get the shortest sequence of commands to pass. If there are multiple solutions, get the lexical minimum one.
To simplify the input, a cell is encoded to an integer as follows. The lowest 4 bits signal the existence of the barriers around a cell. The fifth bit indicates whether a cell is a hole or not. The sixth and seventh bits are set for the start cell and end cell. Details are listed in the following table with bits counted from lowest bit. For a barrier, both of the two adjacent cells will have the corresponding barrier bit set. Note that the first two mazes in the sample input is the encoding of two mazes in the figure above, make sure you understand the encoding right.



| Bit | Value | Description |
|---|---|---|
| 1 | 0 | No barrier to the left |
|  | 1 | A barrier to the left |
| 2 | 0 | No barrier to the bottom |
|  | 1 | A barrier to the bottom |
| 3 | 0 | No barrier to the right |
|  | 1 | A barrier to the right |
| 4 | 0 | No barrier to the up |
|  | 1 | A barrier to the up |
| 5 | 0 | A hole |
|  | 1 | A square |
| 6 | 0 | Not start cell |
|  | 1 | A start cell |
| 7 | 0 | Not end cell |
|  | 1 | An end cell |

Input
The first line of input gives the total number of mazes, T (1 < T ≤ 20). Then follow T mazes. Each maze is a 6*6 matrix, representing the encoding of the original maze. There is a blank line between mazes.

Output

For every two consecutive mazes, you should treat them as a double maze and output the answer. So there are actually T - 1 answers. For each double maze, output the shortest sequence of commands to pass. If there are multiple solutions, output the lexicographically minimum one. If there is no way to pass, output -1 instead.

## Sample Input

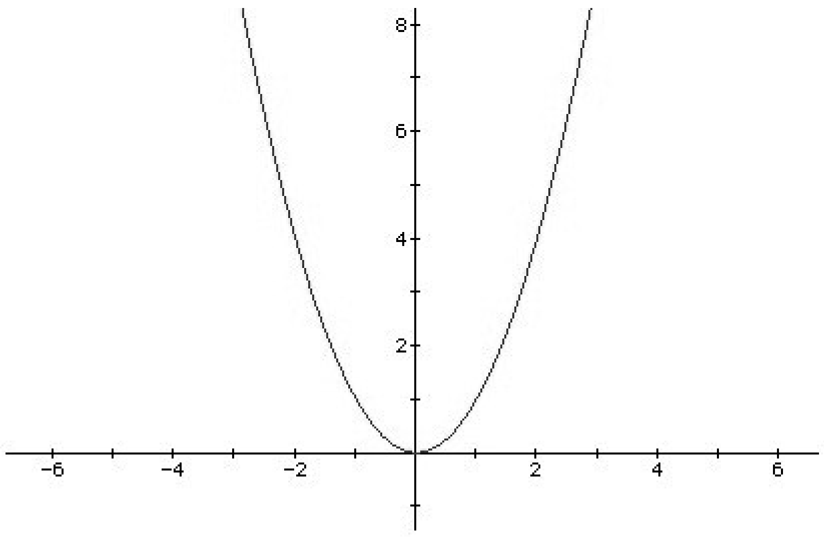
```
3
16 0 18 16 18 24
20 19 24 16 28 1
18 28 17 0 22 17
25 20 17 18 88 20
2 16 48 28 17 16
24 16 16 20 23 1

16 0 18 16 18 24
20 19 24 20 29 1
18 28 17 16 22 17
8 20 1 18 24 20
19 80 48 24 16 0
24 16 16 16 22 19

18 16 18 16 18 80
24 18 24 16 24 18
18 24 0 0 18 24
24 18 0 0 24 18
18 24 18 16 18 24
56 18 24 18 24 18
```

## Sample Output

```
RRLULLLRRDLU
RURDRLLLURDULURRRRRDDU
```

# Error Curves

Problem Description
Josephina is a clever girl and addicted to Machine Learning recently. She pays much attention to a method called Linear Discriminant Analysis, which has many interesting properties.
In order to test the algorithm's efficiency, she collects many datasets. What's more, each data is divided into two parts: training data and test data. She gets the parameters of the model on training data and test the model on test data. To her surprise, she finds each dataset's test error curve is just a parabolic curve. A parabolic curve corresponds to a quadratic function. In mathematics, a quadratic function is a polynomial function of the form f(x) = ax2 + bx + c. The quadratic will degrade to linear function if a = 0.



It's very easy to calculate the minimal error if there is only one test error curve. However, there are several datasets, which means Josephina will obtain many parabolic curves. Josephina wants to get the tuned parameters that make the best performance on all datasets. So she should take all error curves into account, i.e., she has to deal with many quadric functions and make a new error definition to represent the total error. Now, she focuses on the following new function's minimum which related to multiple quadric functions. The new function F(x) is defined as follows: F(x) = max(Si(x)), i = 1...n. The domain of x is [0, 1000]. Si(x) is a quadric function. Josephina wonders the minimum of F(x). Unfortunately, it's too hard for her to solve this problem. As a super programmer, can you help her?

## Input

The input contains multiple test cases. The first line is the number of cases T (T < 100). Each case begins with a number n (n ≤ 10000). Following n lines, each line contains three integers a (0 ≤ a ≤ 100), b (|b| ≤ 5000), c (|c| ≤ 5000), which mean the corresponding coefficients of a quadratic function.

## Output

For each test case, output the answer in a line. Round to 4 digits after the decimal point.

## Sample Input

```
2
1
2 0 0
2
2 0 0
2 -4 2
```

## Sample Output

```
0.0000
0.5000
```

# Go Deeper

Problem Description
Here is a procedure's pseudocode:

```
go(int dep, int n, int m)
begin
output the value of dep.
if dep < m and x[a[dep]] + x[b[dep]] != c[dep] then go(dep + 1, n, m)
end
```

In this code n is an integer. a, b, c and x are 4 arrays of integers. The index of array always starts from 0. Array a and b consist of non-negative integers smaller than n. Array x consists of only 0 and 1. Array c consists of only 0, 1 and 2. The lengths of array a, b and c are m while the length of array x is n. Given the elements of array a, b, and c, when we call the procedure go(0, n, m) what is the maximal possible value the procedure may output?

Input
There are multiple test cases. The first line of input is an integer T ($0 < T \le 100$), indicating the number of test cases. Then T test cases follow. Each case starts with a line of 2 integers n and m ($0 < n \le 200$, $0 < m \le 10000$). Then m lines of 3 integers follow. The i-th($1 \le i \le m$) line of them are $a_{i-1}$ ,$b_{i-1}$ and $c_{i-1}$ ($0 \le a_{i-1}$, $b_{i-1} < n$, $0 \le c_{i-1} \le 2$).

Output
For each test case, output the result in a single line.

Sample Input
```
3
2 1
0 1 0
2 1
```

```
0 0 0
2 2
0 1 0
1 1 2
```

## Sample Output

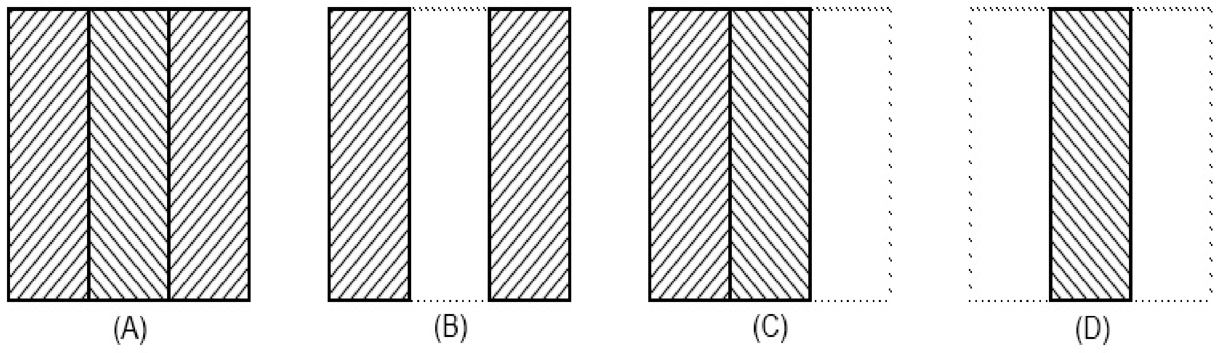
```
1
1
2
```

# Jenga

Problem Description
In their spare time of training, Alice and Charles often play Jenga together. As they've played the game together so many times, they both know each others' performance as well as themselves'. Now with their success rate of each move provided, can you tell in what probability each of them will win? And of course, Alice and Charles, like other ACM-ICPC contestants such as you, are very clever.

Jenga is a game of physical and mental skill. In Jenga, players take turns to remove a block from a tower and balance it on top, creating a taller and increasingly unstable structure as the game progresses.

Jenga is played with 54 wooden blocks. Each block is three times as long as it is wide. To set up the game, the included loading tray is used to stack the initial tower which has 18 levels of three blocks placed adjacent to each other along their long side and perpendicular to the previous level (so, for example, if the blocks in the first level lie lengthwise north-south, the second level blocks will lie east-west).
Once the tower is built, the players take turns to move. Moving in Jenga consists of taking one and only one block from any level (except those mentioned later) of the tower, and placing it on the topmost level in order to complete it. The blocks in the top level, and the level below it if the top level is not completed, cannot be removed. However, if the top level is completed, the blocks in the one below it can be removed. The removed block should be placed to make the top level as same as the other level (with no block removed). The move is successful if the tower does not fall. The game ends when the tower falls, or no block can be removed without making the tower fall (rarely happened). And the loser is the player who made the tower fall (i.e., whose turn it was when the tower fell), or who cannot make the move.



Now let's consider each level of the tower, there're only four types of valid arrangement of wooden blocks, as illustrated above. At the beginning of the game, they're all of the type A (or rotated by 90 degrees). And by removing a block from type A, one will get either type B or type C (or the mirrored equivalent of type C). No block from type B can be removed without making the tower fall. From type C we can only remove a block and result in type D. Then no block can be removed further. So there are only three types of moves: (1) A -> B, (2) A -> C and (3) C -> D, in addition to adding the removed block to the top level.
As Alice and Charles have played Jenga so many times, their success rate of each move is very stable and can be formulated as P = b - d*n, where b is the player's base success rate of this type of move, d is the decrease of success rate for each additional level, and n is the number of levels in the tower before this move. The incomplete top level also counts as one level. For example, if the game begins with 18 levels, and both players have the same performance with b = 2.8 and d = 0.1, then P will be 1.0 for the first turn, and become 0.9 between the 2nd and the 4th turns. If P does not lie in the range [0, 1], the nearest number in the range is indicated. (E.g. when a player cannot fail the first several moves, P will be more than 1 until n is a bit larger.)

## Input

The input file contains multiple test cases. The first line of the input file is a single integer T ($T \le 500$), the number of test cases.
Each test cases begins with a line of $n_0$ ($3 \le n_0 \le 18$), the number of levels in the tower when the game starts. (When $n_0$ is not 18, the rule sare the same.)
The second line contains 6 real numbers $b_{a1}$, $d_{a1}$, $b_{a2}$, $d_{a2}$, $b_{a3}$, $d_{a3}$, indicating Alice's base success rates and the decreases of success rates for each of the three moves: (1) A -> B, (2) A -> C and (3) C -> D. The third line also contains 6 real numbers $b_{c1}$, $d_{c1}$, $b_{c2}$, $d_{c2}$, $b_{c3}$, $d_{c3}$, those of Charles. ($0 \le b - d*n_0 \le 2$ and $0 < d \le 0.5$ for all the 6 pairs of parameters. No real number will have more than 4 digits after the decimal point.)

## Output

For each test case, print a line with Alice's winning probability, assume that she always moves first. Your answer should be rounded to the 4th digit after the decimal point.

## Sample Input

```
2
3
1.3 0.1 1.3 0.1 1.3 0.1
1.3 0.1 1.3 0.1 1.3 0.1
4
1.5 0.1 1.5 0.1 1.5 0.1
1.5 0.1 1.5 0.1 1.5 0.1
```

## Sample Output

```
0.1810
0.8190
```

# Rescue

Problem Description
The princess is trapped in a magic place. In this place, there are N magic stones. In order to rescue the princess, you should destroy all the stones. The N stones are in a straight line. We number them as s1, s2, ... sn from left to right. Each stone has a magic strength $m_1$, $m_2$, ... $m_n$. You have a powerful skill that can do some damage to the stones. To release the skill, you should stand to the right of some stone ($s_i$). Then you throw a power ball towards left. Initially, this ball has a power of p. When it hits a stone, it will do some damage to the stone and its power will be decreased, and the ball will continue to fly left to the next stone if its power is still positive. Formally, if you stand to the right of $s_i$ and the power ball's initial power is p, then the ball will do Max(0, p - (i - j) * (i - j)) damage to sj, for each j <= i. So from this formula, we can see that the damage to stone sj is only determined by the initial power of the ball and the number of stones between $s_i$ and $s_j$. A stone is destroyed if the damage you do is larger than its magic strength. Note that even if a stone is destroyed, it will not disappear; your magic ball will do damage to it and the power will be decreased by that stone. You are not strong enough so that you can release at most k magic balls. It will cost a lot of energy if the power of the magic ball is too high. So what is the minimum value of p with which you can destroy all the magic stones, with no more than k magic balls? You can choose where to release each magic ball as your will, and the power of the ball must be a positive integer.

Input
The first line is the number of cases T (T ≤ 100). For each case, the first line gives two integers n, k (1 ≤ n ≤ 50000, 1 ≤ k ≤ 100000). The second line are n integers, giving $m_1$, $m_2$, ... $m_n$ ($1 \le m \le 10^9$).

Output
Print minimum possible p in a line.

Sample Input

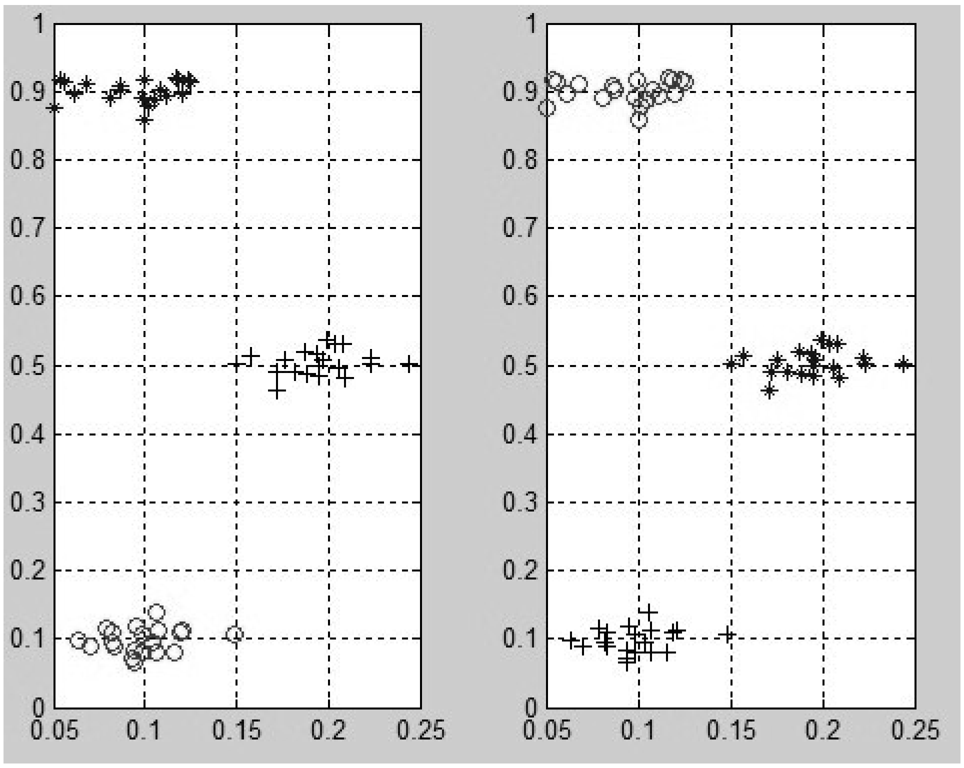
```
2
1 1
1
3 1
1 4 5
```

Sample Output

```
2
6
```

# Similarity

Problem Description
When we were children, we were always asked to do the classification homework. For example, we were given words {Tiger, Panda, Potato, Dog, Tomato, Pea, Apple, Pear, Orange, Mango} and we were required to classify these words into three groups. As you know, the correct classification was {Tiger, Panda, Dog}, {Potato, Tomato, Pea} and {Apple, Pear, Orange, Mango}. We can represent this classification with a mapping sequence{A,A,B,A,B,B,C,C,C,C}, and it means Tiger, Panda, Dog belong to group A, Potato, Tomato, Pea are in the group B, and Apple, Pear, Orange, Mango are in the group C.
But the LABEL of group doesn't make sense and the LABEL is just used to indicate different groups. So the representations {P,P,O,P,O,O,Q,Q,Q,Q} and {E,E,F,E,F,F,W,W,W,W} are equivalent to the original mapping sequence. However, the representations {A,A,A,A,B,B,C,C,C,C} and {D,D,D,D,D,D,G,G,G,G} are not equivalent.



The pupils in class submit their mapping sequences and the teacher should read and grade the homework. The teacher grades the homework by calculating the maximum similarity between pupils' mapping sequences and the answer sequence. The definition of similarity is as follow.

Similarity(S, T) = sum($S_i$ == $T_i$) / L

L = Length(S) = Length(T), i = 1, 2,... L,
where sum($S_i$ == $T_i$) indicates the total number of equal labels in corresponding positions. The maximum similarity means the maximum similarities between S and all equivalent sequences of T, where S is the answer and fixed. Now given all sequences submitted by pupils and the answer sequence, you should calculate the sequences' maximum similarity.

Input
The input contains multiple test cases. The first line is the total number of cases T (T < 15). The following are T blocks. Each block indicates a case. A case begins with three numbers n (0 < n < 10000), k (0 < k < 27), m (0 < m < 30), which are the total number of objects, groups, and students in the class. The next line consists of n labels and each label is in the range [A...Z]. You can assume that the number of different labels in the sequence is exactly k. This sequence represents the answer. The following are m lines, each line contains n labels and each label also is in the range [A...Z]. These m lines represent the m pupils' answer sequences. You can assume that the number of different labels in each sequence doesn't exceed k.

Output
For each test case, output m lines, each line is a floating number (Round to 4 digits after the decimal point). You should output the m answers in the order of the sequences appearance.

Sample Input

```
2
10 3 3
A A B A B B C C C C
F F E F E E D D D D
X X X Y Y Y Y Z Z Z
S T R S T R S T R S
3 2 2
A B A
C D C
F F E
```

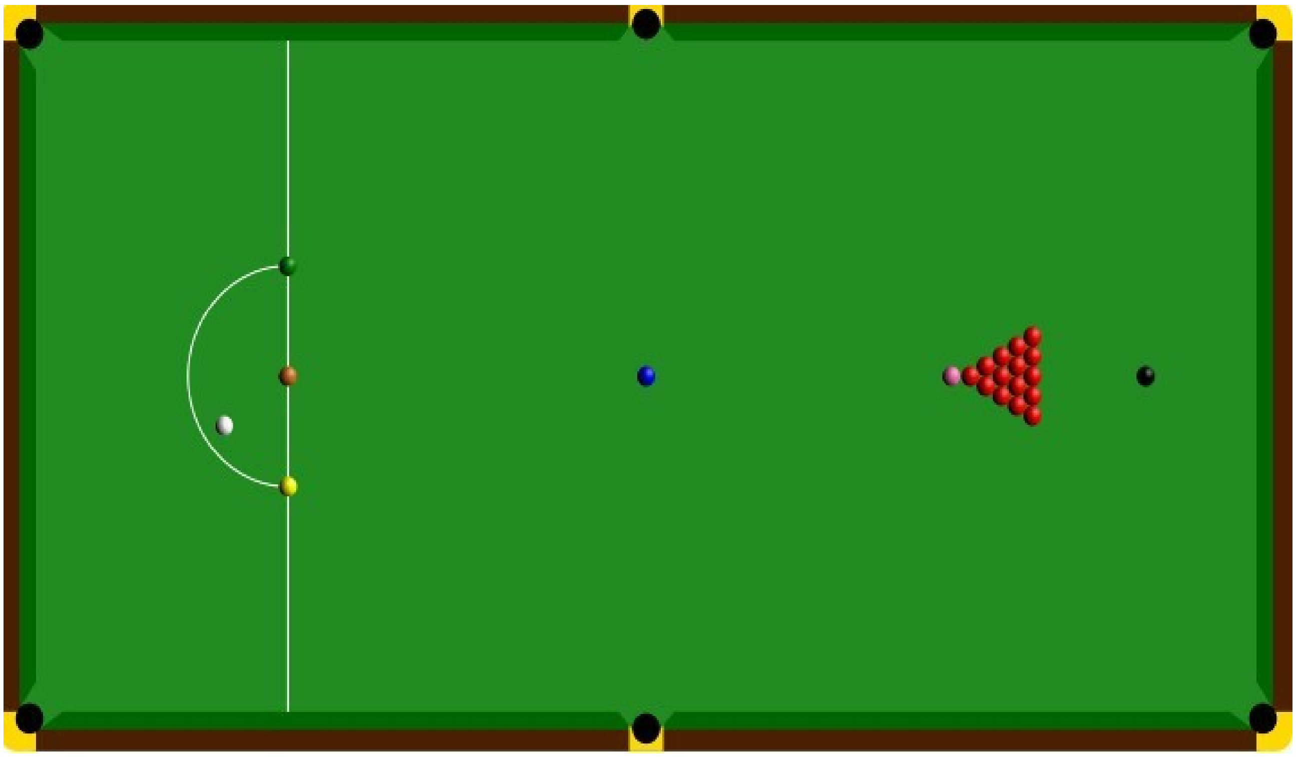
Sample Output

```
1.0000
0.7000
0.5000
1.0000
0.6667
```

# Snooker Referee

Problem Description

Snooker is a cue sport that is played on a large baize-covered table with pockets in each of the four corners and in the middle of each of the long side cushions. It is played using a cue and snooker balls: one white cue ball, 15 red balls worth one point each, and six balls of different colors: yellow (2 points), green (3), brown (4), blue (5), pink (6) and black (7). A player (or team) wins a frame (individual game) of snooker by scoring more points than the opponent(s), using the cue ball to pot the red and colored balls. In this problem, your job is the referee of snooker. You should score both players, ask the correct player to play next, as well as place some of the balls back to the table if necessary. The rules of snooker needed for this problem are following. (We ignore some fouls about incorrectly hitting the cue ball here. We assume that the cue ball is never snookered after a foul, so free ball will never occur. We also assume that both players will make their best attempts to hit the ball on, so you do not need to declare a miss when they do not hit the ball on first.)



At the beginning of each frame the balls are set up by the referee as illustrated above. This will be followed by a break-off shot, the white cue ball can be placed anywhere inside the D (it is called in-hand, which also happens when the cue ball is potted). Players take turns in visiting the table. A break is the number of points scored by a player in one single visit to the table. A player's turn and break end when he fails to pot a ball, when he does something against the rules of the game, which is called a foul, or when a frame has ended.

The ball or balls that can be hit first by the white are called the ball(s) "on" for that particular stroke. The ball(s) "on" differ from shot to shot: a red ball, if potted, must be followed by a color, and so on until a break ends; if a red is not potted, any red ball remains the ball "on". Only a ball or balls "on" may be potted legally by a player. If a ball not "on" is potted, this is a foul.

The game of snooker generally consists of two phases. The first phase is the situation in which there are still red balls on the table. In the first phase, at the beginning of a player's turn, the balls "on" are all remaining red balls. The player must therefore attempt to first hit and pot one or more red balls. For every red ball potted, the player will receive 1 point. When a red has been potted, it will stay off the table and the player can continue his break. If no red has been potted or a foul has been made, the other player will come into play. In case one or more red balls have been potted, the player can continue his break. This time one of the six colors (yellow, green, brown, blue, pink and black) is the ball "on". Only one of these can be the ball "on" and the rules of the game state that a player must nominate his desired color to the referee, although it is often clear which ball the striker is playing and it is not necessary to nominate. When the nominated color is potted, the player will be awarded the correct number of points (yellow, 2; green, 3; brown, 4; blue, 5; pink, 6; black, 7). The color is then taken out of the pocket by the referee and placed on its original spot. Because only one of the colors is the

ball "on", it is a foul to first hit multiple colors at the same time, or pot more than one color. If a player fails to pot a ball "on", it being a red or nominated color, the other player will come into play and the balls "on" are always the reds, as long as there are still reds on the table. The alternation between red balls and colors ends when all reds have been potted and a color is potted after the last red, or a failed attempt to do so is made. Then the second phase begins. All six colors have to be potted in ascending order of their points value (yellow, green, brown, blue, pink, black). Each becomes the ball "on" in that order. During this phase, when potted, the colors stay down and are not replaced on the table, unless a foul is made when potting the color, in which case the color is respotted.
When only the black is left, the first score or foul ends the frame, and the player who has scored most points has won it. However, if the score is tied after that, the black is respotted, the players draw lots for choice of playing, the next player plays from in-hand, and the next score or foul ends the frame.
When a foul is made during a shot, the player's turn is ended and he will receive no points for the foul shot. The other player will receive penalty points. Colors illegally potted are respotted (while reds are not), and if the cue ball is potted, the next player will play from in-hand. Fouls concerned in this problem are:

- failing to hit any other ball with the cue ball
- first hitting a ball "not-on" with the cue ball
- potting a ball "not-on"
- potting the white (in-off)

Penalty points are at least 4 points and at most 7 points. The number of penalty points is the value of the ball "on", or any of the "foul" balls, whichever is highest. When more than one foul is made, the penalty is not the added total - only the most highly valued foul is counted. As players usually do not nominate a color explicitly when hitting the colors, please be tolerant and assume that he always nominate the ball with the lowest score when it cannot be deduced from the ball first hit (i.e. when the cue ball does not hit any ball or hit a red first). If a player commits a foul, and his opponent considers that the position left is unattractive, he may request that the offender play again from the resulting position.

Input
The input file contains multiple test cases. The first line of the input file is a single integer T (T ≤ 200), the number of test cases. For each test case (frame), the first line contains the names of the two players separated by a whitespace. The first player will take the break-off shot. Each name is made up of no more than 20 English letters, and the two names are different.
After that, the input mainly consists of lines that describe a stroke each (with two exceptions stated later). A stroke is described by the color of the ball first hit by the cue ball (or "None" if the cue ball does not hit any ball), followed by zero or more colors of the balls potted, all separated by whitespaces. For example, a line "Red Red White Red" means the cue ball first hit a red ball, and 2 reds are potted as well as the cue ball itself; and a line "None" means the cue ball does not hit any ball thus no ball is potted. You can assume that all strokes are legal according to the balls remain on the table, and the cue

ball will not hit two or more ball first simultaneously. A line "Play again" may appear if and only if the last stroke is a foul. It means the other player request that the offender play again from the resulting position.
If a score or foul occurs when only the black is left, and the score is tied after that, a line with either player's name will follow. That means the player will play next as a result of the lot. The case end when the frame ends. There is a blank line before every test case.

Output
For each frame, print a line in the format "Frame K" first, where K is the index of this case starts from 1. Then use the output to indicate the referee's behaviors:

。When a frame begins, print a line in the format "PlayerName's turn, in-hand", where PlayerName is the name of the player who take the break-off shot.

。After each stroke, print a line "Foul!" first if it is a foul. Then print a line with current score in the format "ScoreA : ScoreB", where ScoreA is the score of the player who take the break-off shot, and ScoreB is the other player's score.
After that, if the frame continue fairly (i.e. not only black is
left before the stroke, or it is not a score or foul when only black is left), and some ball(s) should be respotted. Print a line with the word "Respot" following by the color(s) of the ball(s), all separated by whitespace. If more than one ball should be respotted, print their colors in ascending order of their values. Do not print anything if no ball needed to be respotted.
At last (when the frame continue fairly and any necessary ball has been respotted), if the last player's break ends, print a line in the format "PlayerName's turn" to ask the other player to play next, where PlayerName is the next player's name. If the next player should play from in-hand, print "PlayerName's turn, in-hand" instead.

。After a foul, if the other player request that the offender play again, just print a line "PlayerName's turn" or "PlayerName's turn, in-hand" according to whether the cue ball is in-hand, where PlayerName is the offender's name. Note that the requester is actually requesting in his turn, after you asked him to play next.

。If a score or foul occurs when only the black is left, and the score is tied after that, print a line "Tie" after the score. Then print two lines "Respot Black" and "PlayerName's turn, in-hand" to respot the black and play from in-hand, where PlayerName is the next player's name (determined by the lot).

。When the frame ends, print a line "PlayerName wins" after the score, where PlayerName is the winner's name.

Print a blank line between every two successive cases.

Sample Input
```
1
Zero Maxbreak
```

```
Red White
Red Red
Black Black
Red Red
Black Black
Red Red
Black Black
Red Red
Black Black
Red Red
Black Black
Red Red
Black Black
Red Red
Black Black
Red Red
Black Black
Red Red
Black Black
Red Red
Black Black
Red Red
Black Black
Red Red
Black Black
```

## Sample Output

```
Frame 1
Zero's turn, in-hand
Foul!
0 : 4
Maxbreak's turn, in-hand
0 : 5
0 : 12
Respot Black
0 : 13
0 : 20
Respot Black
0 : 21
0 : 28
Respot Black
0 : 29
0 : 36
Respot Black
0 : 37
0 : 44
Respot Black
0 : 45
0 : 52
Respot Black
0 : 53
0 : 60
Respot Black
0 : 61
0 : 68
```