Problem A. Trainsorting

Input file:	`trainsorting.in`
Output file:	`trainsorting.out`

Erin is an engineer. She drives trains. She also arranges the cars within each train. She prefers to put the cars in decreasing order of weight, with the heaviest car at the front of the train.

Unfortunately, sorting train cars is not easy. One cannot simply pick up a car and place it somewhere else. It is impractical to insert a car within an existing train. A car may only be added to the beginning and end of the train.

Cars arrive at the train station in a predetermined order. When each car arrives, Erin can add it to the beginning or end of her train, or refuse to add it at all. The resulting train should be as long as possible, but the cars within it must be ordered by weight.

Given the weights of the cars in the order in which they arrive, what is the longest train that Erin can make?

Input

The first line contains an integer $0 \leq n \leq 2000$, the number of cars. Each of the following n lines contains a non-negative integer giving the weight of a car. No two cars have the same weight.

Output

Output a single integer giving the number of cars in the longest train that can be made with the given restrictions.

Example

`trainsorting.in`	`trainsorting.out`
3 1 2 3	3

Problem B. Hot Spot

Input file:	`hotspot.in`
Output file:	`hotspot.out`

Hot Spot is a single player game played on a 4 by 4 game board. The purpose of the game is to move a red robot from its current location on the board to the top left corner. The game board may also contain green and blue robots. Each square of the game board can be occupied by no more than one robot at any time.

A robot may move in one of two ways:

1. If two robots are adjacent horizontally or vertically (but not diagonally), one of them may jump over the other to the immediately adjacent square, provided that square is empty. For example, if robot a is immediately to the left of robot b, robot a may jump to the square immediately to the right of robot b.
2. If three robots are adjacent in a line (again not diagonally), one of them may jump over the other two, provided the destination square is empty. For example, if robot b is immediately to the right of robot a and robot c is immediately to the right of robot b, robot a may jump to the square immediately to the right of robot c.

Every jump only changes the positions of the existing robots; robots are never removed from or added to the game board.

A blue robot is never allowed to be adjacent horizontally or vertically to another blue robot or to the red robot.

Given the initial configuration of the game board, determine the minimum number of jumps required to move the red robot into the top left corner.

Input

The input specifies the initial position of the board using four lines, each containing four characters. Each character may be either R, indicating the red robot, B, indicating a blue robot, G, indicating a green robot, or a period (.), indicating an empty square.

Output

Output a single line containing a single integer, the minimum number of jumps required for the red robot to reach the top left square of the game board.

Example

`hotspot.in`	`hotspot.out`

`.GR.`
`....`
`....`
`....`
	1

Problem C. Snakes and Ladders

Input file:	`snakesandladders.in`
Output file:	`snakesandladders.out`

Snakes and Ladders is a board game played on a 10 by 10 grid. The squares of the grid are numbered 1 to 100. Each player has a token which is placed on the square numbered 1 at the beginning of the game. Players take turns rolling a die which provides a random number between 1 and 6. After rolling, the player advances his or her token the number of squares shown on the die. If this would put the token past the square numbered 100, the token is advanced to 100. After advancing, if the token is on a square containing the bottom of a ladder, the token must be moved to the square containing the top of the ladder. Similarly, if the token is on a square containing the mouth of a snake, the token must be moved to the square containing the tail of the snake. No square contains more than one endpoint of any snake or ladder. The token numbered 100 does not contain the mouth of a snake or the bottom of a ladder. A player wins when his or her token reached the square numbered 100. At that point, the game ends.

Given the configuration of the snakes and ladders on a game board and a sequence of die rolls, you are to determine the positions of all the tokens on the game board. The sequence of die rolls need not be complete (i.e. it need not lead to any player winning). The sequence of die rolls may also continue after the game is over; in this case, the die rolls after the end of the game should be ignored.

Input

The first line contains three positive integers: the number a of players, the number b of snakes or ladders, and the number c of die rolls. There will be no more than 1000000 players and no more than 1000000 die rolls. Each of the next b lines contains two integers specifying a snake or ladder. The first integer indicates the square containing the mouth of the snake or the bottom of the ladder. The second integer indicates the square containing the tail of the snake or the top of the ladder. The following c lines each contain one integer giving number rolled on the die.

Output

For each player, output a single line containing a string of the form "`Position of player N is P.`", where N is replaced with the number of the player and P is replaced with the final position of the player.

Example

snakesandladders.in

3 1 3
4 20
3
4
5

snakesandladders.out
Position of player 1 is 20. Position of player 2 is 5. Position of player 3 is 6.

Problem D. Virtual Friends

Input file: `virtualfriends.in`
Output file: `virtualfriends.out`

These days, you can do all sorts of things online. For example, you can use various websites to make virtual friends. For some people, growing their social network (their friends, their friends' friends, their friends' friends' friends, and so on), has become an addictive hobby. Just as some people collect stamps, other people collect virtual friends.

Your task is to observe the interactions on such a website and keep track of the size of each person's network.

Assume that every friendship is mutual. If Fred is Barney's friend, then Barney is also Fred's friend.

Input

The first line of input contains one integer specifying the number of test cases to follow. Each test case begins with a line containing an integer F, the number of friendships formed, which is no more than 100 000. Each of the following F lines contains the names of two people who have just become friends, separated by a space. A name is a string of 1 to 20 letters (uppercase or lowercase).

Output

Whenever a friendship is formed, print a line containing one integer, the number of people in the social network of the two people who have just become friends.

Example

virtualfriends.in	virtualfriends.out
1 3 Fred Barney Barney Betty Betty Wilma	2 3 4

Problem E. Dominos

Input file: `dominos.in`
Output file: `dominos.out`

Dominos are lots of fun. Children like to stand the tiles on their side in long lines. When one domino falls, it knocks down the next one, which knocks down the one after that, all the way down the line. However, sometimes a domino fails to knock the next one down. In that case, we have to knock it down by hand to get the dominos falling again.

Your task is to determine, given the layout of some domino tiles, the minimum number of dominos that must be knocked down by hand in order for all of the dominos to fall.

Input

The first line of input contains one integer specifying the number of test cases to follow. Each test case begins with a line containing two integers, each no larger than 100 000. The first integer n is the number of domino tiles and the second integer m is the number of lines to follow in the test case. The domino tiles are numbered from 1 to n. Each of the following lines contains two integers x and y indicating that if domino number x falls, it will cause domino number y to fall as well.

Output

For each test case, output a line containing one integer, the minimum number of dominos that must be knocked over by hand in order for all the dominos to fall.

Example

dominos.in	dominos.out

1
3 2
1 2
2 3
	1

Problem F. Logo

Input file: `logo.in`
Output file: `logo.out`

Logo is a programming language built around a turtle. Commands in the language cause the turtle to move. The turtle has a pen attached to it. As the turtle moves, it draw lines on the page. The turtle can be programmed to draw interesting pictures.

We are interested in making the turtle draw a picture, then return to the point that it started from. For example, we could give the turtle the following program:

```
fd 100 lt 120 fd 100 lt 120 fd 100
```

The command `fd` causes the turtle to move forward by the specified number of units. The command `lt` causes the turtle to turn left by the specified number of degrees. Thus the above commands cause the turtle to draw an equilateral triangle with sides 100 units long. Notice that after executing the commands, the turtle ends up in the same place as it started. The turtle understands two additional commands. The command `bk` causes the turtle to move backward by the specified number of units. The command `rt` causes the turtle to turn right by the specified number of degrees.

After executing many commands, the turtle can get lost, far away from its starting position. Your task is to determine the straight-line distance from the turtle's position at the end of its journey back to the position that it started from.

Input

The first line of input contains one integer specifying the number of test cases to follow. Each test case starts with a line containing one integer, the number of commands to follow. The commands follow, one on each line. Each test case will contain no more than 1000 commands.

Output

For each test case, output a line containing a single integer, the distance rounded to the nearest unit.

Example

logo.in	logo.out
1 5 fd 100 lt 120 fd 100 lt 120 fd 100	0

Problem G. Cranes

Input file: `cranes.in`
Output file: `cranes.out`

A crane is a wonderful tool for putting up a building. It makes the job go very quickly. When the building must go up even faster, more than one crane can be used. However, when there are too many cranes working on the same building, it can get dangerous. As the cranes spins around, it can bump into another crane if the operator is not careful. Such an accident could cause the cranes to fall over, possibly causing major damage. Therefore, safety regulations require cranes to be spaced far enough apart so that it is impossible for any part of a crane to touch any part of any other crane. Unfortunately, these regulations limit the number of cranes that can be used on the construction site, slowing down the pace of construction. Your task is to place the cranes on the construction site while respecting the safety regulations.

The construction site is laid out as a square grid. Several places on the grid have been marked as possible crane locations. The arm of each crane has a certain length r, and can rotate around the location of the crane. The crane covers the entire area that is no more than r units away from the location of the crane. You are to place the cranes to maximize the total area covered by all the cranes.

Input

The first line of input contains one integer specifying the number of test cases to follow. Each test case begins with a line containing an integer C, the number of possible locations where a crane could be placed. There will be no more than 15 such locations. Each of the following C lines contains three integers x, y, and r, all between $-10\,000$ and $10\,000$ inclusive. The first two integers are the grid coordinates of the location, and the third integer is the length of the arm of the crane that can be placed at that location.

Output

For each test case, find the maximum area A that can be covered by cranes, and output a line containing a single integer B such that $A = B \times \pi$.

Example

cranes.in	cranes.out

1
3
0 0 4
5 0 4
-5 0 4
	32

Problem H. WiFi

Input file: `wifi.in`
Output file: `wifi.out`

One day, the residents of Main Street got together and decided that they would install wireless internet on their street, with coverage for every house. Now they need your help to decide where they should place the wireless access points. They would like to have as strong a signal as possible in every house, but they have only a limited budget for purchasing access points. They would like to place the available access points so that the maximum distance between any house and the access point closest to it is as small as possible.

Main Street is a perfectly straight road. The street number of each house is the number of metres from the end of the street to the house. For example, the house at address 123 Main Street is exactly 123 metres from the end of the street.

Input

The first line of input contains an integer specifying the number of test cases to follow. The first line of each test case contains two positive integers n, the number of access points that the residents can buy, and m, the number of houses on Main Street. The following m lines contain the house numbers of the houses on Main Street, one house number on each line. There will be no more than 100 000 houses on Main Street, and the house numbers will be no larger than one million.

Output

For each test case, output a line containing one number, the maximum distance between any house and the access point nearest to it. Round the number to the nearest tenth of a metre, and output it with exactly one digit after the decimal point.

Example

wifi.in	wifi.out

1
2 3
1
3
10
	1.0

Problem I. Logo 2

Input file: `logo2.in`
Output file: `logo2.out`

Logo is a programming language built around a turtle. Commands in the language cause the turtle to move. The turtle has a pen attached to it. As the turtle moves, it draw lines on the page. The turtle can be programmed to draw interesting pictures.

We are interested in making the turtle draw a picture, then return to the point that it started from. For example, we could give the turtle the following program:

```
fd 100 lt 120 fd 100 lt 120 fd 100
```

The command `fd` causes the turtle to move forward by the specified number of units. The command `lt` causes the turtle to turn left by the specified number of degrees. Thus the above commands cause the turtle to draw an equilateral triangle with sides 100 units long. Notice that after executing the commands, the turtle ends up in the same place as it started. The turtle understands two additional commands. The command `bk` causes the turtle to move backward by the specified number of units. The command `rt` causes the turtle to turn right by the specified number of degrees. The distances and angles in all commands are always non-negative integers.

Unfortunately, we have been messy in writing the program down, and cannot read our own writing. One of the numbers in the program is missing. Assuming the turtle ends up at the place that it started at the end of its journey, can you find the missing number?

Input

The first line of input contains one integer specifying the number of test cases to follow. Each test case starts with a line containing one integer, the number of commands to follow. The commands follow, one on each line. Each test case will contain no more than 1000 commands. The argument of each command is either an integer or a question mark (`?`). There will be exactly one question mark in each test case.

Output

For each test case, output line containing a single integer n such that when the question mark in the program is replaced by n, the turtle ends up at the same point that it started from once the program completes. If the question mark is the argument of an `lt` or `rt` command, the angle in the output must be between 0 and 359 degrees, inclusive. The correct answer will always be an integer, and we guarantee that for every test case, there will be only one correct answer.

Example

logo2.in	logo2.out

`1`
`5`
`fd 100`
`lt 120`
`fd ?`
`lt 120`
`fd 100`
	`100`