BALDWIN WALLACE UNIVERSITY

2014 HIGH SCHOOL PROGRAMMING CONTEST

DO NOT OPEN UNTIL INSTRUCTED TO DO SO

These Are Not The CAPS You're Looking For

During Luke's mission to rescue Princess Leia and unknown to the Empire, R2D2 hacked into the communications center in the Death Star. During this time, R2D2 planted a bug that would allow a very simple translation of text to slip through their data center and allow the Rebels to send transmissions secretly. The encryption is purely syntactical: every other lowercase letter, starting with the first lowercase letter, is changed to uppercase. All whitespace, punctuation, and uppercase letters should not be altered. You are to write a program to encrypt all traffic so that it slips through the Death Star's data center. May the force be with you!

Details of input:

The first line of input will contain a positive integer *n* indicating the number of cases to be tested (*n* <= 50). Each of the following *n* lines will contain a string to be capitalized.

Details of output:

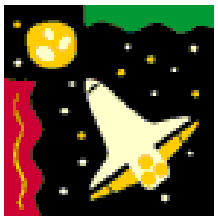
For each case you are to output the encrypted string.

Sample Input:

```
2
Empire Sux! Go Rebels!
check for traps!!!!1
```

Sample Output:

```
EMpIrE SuX! Go REbElS!
ChEcK fOr TrApS!!!!1
```



Shoot the Death Star!

Behold! The Death Star approaches near as your ship zooms through the intergalactic darkness. You enter three settings to aim your cannons, then fire! The missiles soar through the black for what seems like hours but do they hit? Your eyes are fixated on the evil mass.

Write a program that determines whether or not the missile hits the Death Star based on the 3 calibration settings: *x*, *y*, and *z*. The input will consist of several test cases. The first line of input will contain a single positive integer, *n*, indicating the number of test cases to follow. Each of the following *n* lines contains a single test case. Each line contains 3 integers, separated by a single space. The first integer is *x*, followed by *y*, and then *z*. *x* and *y* will be non-negative and *z* will never be 0. The location of the missile can be determined by the expression

((x*3)-y)/z

If the resulting value is strictly greater than 100 and strictly less than 300, the test case is a success. Otherwise, it is a failure. Your output should show the results of the test cases, each on a separate line.

Successful cases will respond with the output:

You have hit the Death Star!

Unsuccessful cases will respond with:

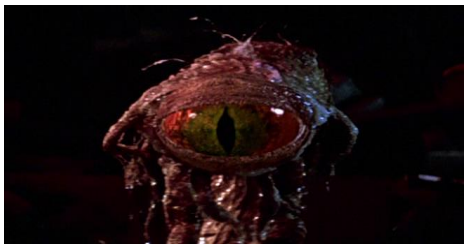
You have missed the Death Star.

Sample input:

```
3
100 50 2
100 200 50
300 60 4
```

Sample output:

```
You have hit the Death Star!
You have missed the Death Star.
You have hit the Death Star!
```



Revenge of the Dianoga

There is a dianoga swimming in your garbage compactor! He is controlled directly by Lord Sidious. Lord Sidious wants him to grow as big as possible so that Luke, Han, Leia, and Chewey can't escape next time. Lord Sidious gives the dianoga directions so that he eats as much food as possible, while also avoiding traps and the walls of the garbage compactor. If the dianoga eats a piece of food, its tail grows by one foot (which is equal to one space on your game board). If it hits a trap or a wall it dies!

Well… Lord Sidious got thrown down that big scary blue tube…. so we need a new way to control this beast. You will write a program to do this! You must also report back to your superiors whether or not the dianoga dies.

Details of the Input:
The first line will contain a single integer *n* which represents the number of test cases. Each test case will contain three sections. The first line in each test case will contain two positive integers, $0 < h <= 15$ (representing the height of the game board) and $0 < w <= 15$ (representing the width of the game board). Following that there will be *h* lines with *w* characters on each line. Each character can be one of 4 things:

s = the beginning/head of the snake
f = food
t = trap
- = empty space

There will only be one 's' on the board and the snake will always start with length of 1. There can be multiple 'f' and 't' on the board. After the dianoga eats a food, the food is removed from the board.

The last line of each test case will contain a positive integer $0 < d < 50$ representing the number of commands that your dianoga will follow. Following this there will be *d* characters representing the direction in which the dianoga will move. The possible values for directions are *n, s, e, w* representing *north*, *south*, *east*, and *west* respectively.

Details of the Output:
You should output one of the following:

It's a trap!	*if the dianoga hit a trap*
The dianoga hit the *y* wall!	*where y is north, south, east, or west*
The dianoga bit itself!	*if the dianoga ran into itself*
Path complete!	*if all commands are completed*

On the same line, report:

Dianoga length is *x*.	*where x is a positive integer*

Sample input

```
2
8 8
--------
--------
--------
--------
---sf---
--------
--------
--------
6 eeeeee
10 10
----------
----------
----------
----------
---------t
---------f
---------f
---------f
---------f
---------s
8 nnnnnnnn
```

Sample output

```
The dianoga hit the east wall! Dianoga length is 2.
It’s a trap! Dianoga length is 5.
```

Build a Lightsaber

You are an up and coming Jedi and you must craft your own lightsaber. Your master has taught you how to determine the power of a lightsaber, and you wish to create a program to calculate this. A lightsaber has many parts, each with its own power level. The power of the lightsaber is then defined as the greatest common divisor of these powers.

The greatest common divisor of a set of nonzero integers $\{X_1, X_2, \ldots, X_n\}$ is defined as the largest positive integer N such that N divides $X_1, X_2, \ldots, X_n$ without remainder.

Details of the Input
The first line of input will be a positive integer *n* indicating the number of cases to be tested. Each case will begin with a positive integer *k* ($1 \leq k \leq 50$) indicating how many integers are in the set. Following this, on the same line, there will be *k* nonzero integers. You are to find the greatest common divisor of these integers.

Details of the Output
For each test case you are to output the greatest common divisor of the set of integers.

Sample Input
```
2
3 7 5 9
4 585 200 2405 50
```

Sample Output
```
1
5
```

Jabba the Hutt's Necklace Hut

Little known fact: before his role in the Star Wars movies, Jabba owned a small necklace hut. He sold a large variety of necklaces, but they all had special qualities that made them clearly from Jabba the Hutt's Necklace Hut:

1. Each necklace consists of some number, *n*, of colored sections.
2. Each color will only appear in one section.
3. Each section only has one color in it.

For example, a sample necklace with the colors Red, Orange, and Yellow in that order would be Red-Orange-Yellow-Red (we signify the red a second time to mark that it has completed a circle).

However, a rival neckwear business (Leia's Leis) has sent a saboteur to break all of Jabba's supplies! Each necklace was broken into multiple pieces that Jabba found lying on the floor and doesn't know how to put them back together again. It's up to you to fix them!

Fortunately, you notice that each necklace was broken in the same way. Remember that each necklace has *n* colors. Well, it turns out that each necklace was then broken into *n* parts, each of which contains two colors. As such, we can represent a part by the two colors it consists of, for example: RG would represent a part that was green on one half and red on the other. It's up to you to figure out what each necklace looked like originally.

Details of the Input:
The first line of input will be a single integer, *d*, giving the number of data sets. Each subsequent line will contain an integer, *n*, ($1<n<100$). That number will be followed by *n* 'necklace parts' which each consist of two different characters. Valid characters will only be A-Z.

Details of the Output:
Each line of the output will contain the finished necklace starting with the first piece in the input facing that direction. For instance, if the first piece given was RG, then the necklace's output *must* begin with the characters RG. The final necklace should be output with no spaces and should start and end with the same character. Each character besides this one should only appear once in the output line.

Sample Input:

```
3
5 RW GO OR YG WY
3 GR RO GO
8 SW OR DS OW ER PI PE ID
```

Sample Output:

```
RWYGOR
GROG
SWOREPIDS
```

Versus Vader

Darth Vader has challenged you to a game. The stakes? Your home planet.

Fortunately, he was a nice guy about it and gave you the rules to the game ahead of time. An official will give a multi-digit number. You then perform certain operations on this number to try to maximize your score. You and your opponent do your calculations separately and then compare results at the end. Whoever has the largest score gets to decide your planet's fate.

You start by taking a single digit from either end of the number and score that many points. On the second move, you remove 2 digits from either end. On each successive turn, the number of digits removed grows by one (3, 4, etc.) but they can always be taken from either end. This will continue until all of the original number has been consumed. You know that the number will be such a length that by following these rules, you will be able to remove the appropriate number of digits on each turn, eventually obtaining a value of 0. You also know that the largest number of digits you will remove will be 8.

Your goal is to write a program to find the highest possible score so you know that Vader will never get a better score than you.

Details of the input:
The first line will consist of a single integer, *n*, telling how many test games you will play. Each of the next *n* lines will hold a single multi-digit number containing no more than 36 digits.

Details of the output:
On each line, give the maximum score that can be achieved for the given input.

Sample input:
```
3
123456
5
215
```

Sample output:
```
480
5
26
```

Out of Gas

You're a huge fan of all sci-fi movies. But your favorite of all time has to be Star Wars. You've seen those three movies hundreds of times. That's why when someone casually mentioned to you that there was a prequel trilogy you got psyched to watch them! You popped in Episode I. However, after watching such a train wreck of a movie, you realize your obligation to your fandom. You successfully gathered all evidence of a prequel trilogy in the middle of a desert and have rigged an explosion to blow in 30 minutes. Unfortunately, your electric car, the Firefly, that you drove to the desert is now out of battery power and you'll have to rely solely on the random cans of gas scattered around the desert to escape with your life.

Your goal is to figure out how far away from your starting location you can get in just 30 minutes. You start with a full can of gas, but a single can of gas is not very large and you will run out before you get to safety. Fortunately, there just happen to be full gas canisters scattered around the desert that you can refuel with. When you arrive at a gas canister, it instantly fills your car. That is, no time is spent in the refilling process and your car is always full as you move away from the canister. Once a gas can is used, it cannot be used again even if you return to the same location. If necessary, you can also get out of your car, walk to one of the gas cans and bring it back to your car. As soon as you get back to your car, your tank instantly is filled and you can drive on.

Your escape strategy must follow some specific rules:

- All time is measured by how long it takes you to move 1 unit either by driving or walking, times which may differ from each other.
- When you use your car to drive one unit, you use one unit of gas.
- You cannot move diagonally, only up, down, left, and right.
- You cannot move half of a unit. For instance, if it takes you 2 minutes to move 1 unit and you only have 1 minute left, you can't move half a unit. As such, you may end up with time leftover, but unable to take any actions.

You will determine the farthest position that you can be from the explosion. You position is determined in the following way:

- Every location is viewed as an ordered pair in a coordinate plane relative to where you started. That is, the place where you set the explosive the origin (0,0) of the coordinate system.
- Positive distance is up or to the right relative to the origin. Negative distance is down or to the left relative to the origin. For example, if you finish 10 units up and two units left of the origin, your final location is (10,-2).
- Your total distance from the explosion is defined as the sum of the magnitude of the distances in the X and Y direction. For the coordinate (10, -2) the total distance would be 12 units.
- While you like your car, you won't risk your life to try to save it, so you may end up not in your car at the end. If so, treat the final distance as your location, not the car's location.

Details of the input:
The first line of input is a positive integer, *n*, that indicates the number of test cases. For each test case, the first line of input will have the number of gas cans, *g*, located in the desert. The following *g* lines will each contain two integers that represent the coordinates where the gas cans are located. The next line will have a single positive integer, *f*, which indicates the amount of gas you holds when the tank is full (NOTE: you start with a full tank of gas). The final line of input for each case will be two positive integers, *d* and *w*, indicating how many minutes it takes you to drive one unit and walk one unit, respectively.

Details of the output:
For each input case, output the maximum distance you can be away from your starting location.

Sample input:
```
2
3
2 2
1 1
-3 1
4
3 10
2
20 5
-3 -2
2
10 15
```

Sample output:
```
8
2
```