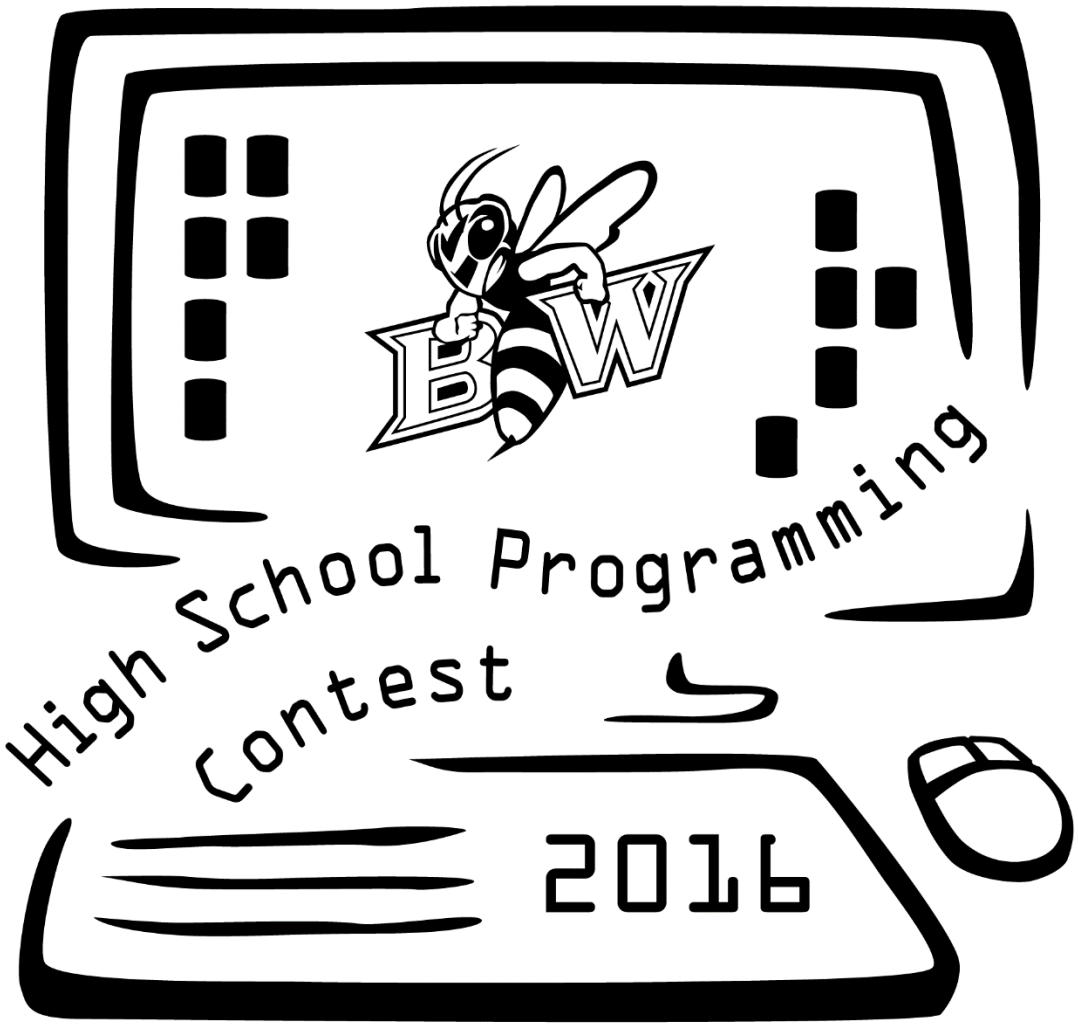
BALDWIN WALLACE UNIVERSITY

2016 HIGH SCHOOL PROGRAMMING CONTEST



DO NOT OPEN UNTIL INSTRUCTED TO DO SO

The Last Human Left

The children of Winterfell love to play a game called Human-White Walker. The game begins with all the players arranged in a circle. One of the children is randomly chosen to start the game. The starter taps the child directly on his/her left and that child turns into a White Walker and is out of the game. The next child to the left now becomes the tapper and taps the child to her/his left. Again, the tapped player turns into a White Walker and is out. Each time someone leaves the game, the circle moves inward to keep everyone side by side. Play continues in this way until a single child remains – the winner!

As an example, let's say that there are 5 children playing the game. Let's call them Child 0 (the starter), Child 1, Child 2, Child 3, and Child 4. Child 0 taps Child 1 (who is out), then Child 2 taps Child 3 (who is out). Child 0, Child 2, and Child 4 remain in the circle. As the turn moves to the left, Child 4 taps Child 0. Now this leaves only Child 2 and Child 4. Since it is Child 2's turn, they tap Child 4, turning him/her into a White Walker and leaving Child 2 as the only remaining Human and winner of the game.

You are going to simulate the Human-White Walker game for different starting sizes of circles and determine who will win the game.

Details of the input
The input begins with a line containing a single integer, c, that is the number of games you will simulate. For each case, there is a single line of input containing an integer n, ($1 \leq n \leq 1000$), which represents the starting size of the circle.

Details of the output
You will output one line for each case. The line begins with a label of the form "`Case i:`" ($i = 1...c$) and then prints an integer that is the original number of the child that is the winner.

Sample input
```
2
5
10
```

Sample output
```
Case 1: 2
Case 2: 4
```

Working at the Wall

You are responsible for recruiting men to work for the Night's Watch. You decide to determine the number of men you need to recruit each evening by way of a modified version of the Fibonacci sequence. The Fibonacci sequence is the following integer sequence constructed by adding the two previous terms of the sequence.

0, 1, 1, 2, 3, 5, 8, 13, 21, 33, 54, 89 …

That is, the n^{th} Fibonacci term can be written as $F_n = F_{n-1} + F_{n-2}$, where $F_0 = 0$ and $F_1 = 1$. If the initial values of F_0 and F_1 are different than 0 and 1, a different sequence of number would be generated.

For instance, if we start with $F_0 = 0$ and $F_1 = 3$, F_2 is found the following way:

$$F_2 = F_1 + F_0 = 3 + 0 = 3$$

F_3 found the following way:

$$F_3 = F_2 + F_1 = 3 + 3 = 6$$

And so on….

The number of workers that will be used for Night's Watch will be generated using this approach. For each evening, values for F_0 and F_1 will be specified as will which term will determine the number of workers.

Details of the input

The first line of input will contain a single integer, *c*, that is the number of cases to be solved. For each case, there will be a single line of input containing three integers *f, s,* and *n,* ($0 \leq f, s \leq 1000$, $2 \leq n \leq 50$), with *f* and *s* specifying the first and second values of the sequence and *n* specifies the term that determines the number of workers.

Details of the output

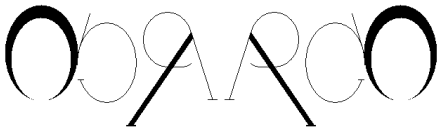
You will output one line for each case. The line begins with a label of the form "`Case i:`" (*i* = 1...*c*) and then prints an integer that is the number of workers for that case.

Sample input

```
2
0 3 5
2 4 10
```

Sample output

```
Case 1: 15
Case 2: 288
```



What Can Hodor Say?

Hodor, a strong, large man is only capable of saying one word: "Hodor". Understandably, this makes it difficult for him to communicate with others. However, he recently discovered that, with a lot of practice, he is able to say another type of word – palindromes. These are words that are spelled the same way when read from left to right and when read from right to left. For example, "mom" and "racecar" are both palindromes.

Hodor discovered this ability when he was reading an old book about dragons. He has a large collection of similar books and would like to know which book has the greatest number of palindromes for him to practice. He has asked for your help in analyzing the text of the books to count palindromes. However, there are a few requirements you must follow:

1. Anything that is not a letter (either uppercase or lowercase) should be ignored. For example, the following would be considered a palindrome because the space is ignored: "hee h".
2. The palindromes must be at least two letters long. A single letter is not considered a palindrome.
3. Case does not matter. "kAyaK" is still a palindrome even though the letters are not all the same case.
4. Palindromes that are embedded in larger words still count. For example, the word "automotive" as a whole is not a palindrome, but contained within the word is the palindrome "omo" and "tomot".
5. The palindromes do not need to be real words and they do not need to be unique.

Details of the input

The first line of the input will be a positive integer, *n*, which gives the number of test cases to follow. The following *n* lines each contain some combination of uppercase and lowercase letters, spaces, and various punctuation.

Details of the output

You will output one line for each case. The line begins with a label of the form "`Case i:`" (*i* = 1...*c*). Following the label the output will say: "`There are [value] palindromes in that book.`" with `[value]` replaced by the number (an integer) of palindromes you have found on that line.

Sample input

```
1
HeLlo theRe, how.arE you?
```

Sample output

```
Case 1: There are 3 palindromes in that book.
```

Talking in Circles

You are working in Storm's End as a spy for King Ares, who doesn't trust House Baratheon. While you are there, you find troubling information about potential plans of Lord Robert Baratheon to overthrow the King. You have enlisted the assistance of a messenger to get word back to Ares of the impending attack, but are concerned that if the message is intercepted along the way, you will be revealed as a spy and punished. So you figure out a clever way to encrypt the message before handing it off to be delivered.

The first step of the encryption is to remove all spaces from the string. The message is then copied into the smallest square grid that can hold all the characters. If the length of the sides of the grid is an odd number, then the first character of the message is placed in the middle cell of the grid and the remainder of the message will be placed in concentric square "rings" working from the inside of the grid out. That is, the sides of the first ring will be of length three, the next of length five, and so on. If the length of the sides of the grid is an even number, then the first ring is comprised of the middle 4 cells of the grid (i.e., a ring whose side are of length 2). As in the other case, the remaining characters are placed in increasingly bigger rings working from the inner part of the grid to its outer edge. In both cases, each ring starts in the top left corner of the ring and works around the square in a clockwise manner. When a ring is finished, a new ring is made around it. If you get to the end of the string and the outer ring is not yet filled, the remaining cells will contain the letter "X".

Details of the input

The first line of input will be a positive integer, *c*, indicating the number of cases. Each of the next *n* lines contains a string consisting of alphanumeric characters (A-Z, 0-9) and spaces. Each string will be between 1 and 100 characters.

Details of the output

For each test case, the first line of output is a label of the form "`Case i:`" where *i* is the current case (1...*c*). The remaining output for each case will be 1 to 10 lines displaying the cipher grid, with each row of the grid on its own line.

Sample input

```
2
PROGRAMMING IS FUN
HELLO
```

Sample output

```
Case 1:
RAMM
NPRI
UGON
FSIG
Case 2:
ELL
XHO
XXX
```

Wights at the Wall

Recent clashes between the Night's Watch and the wildlings have left many bodies to be reanimated as wights near the Wall. As the wights take form, they begin to attack the villages moving from east to west (that is, starting at Whitetree, moving to Craster's Keep and so on). The wildling villagers, wanting no part of these creatures, are attempting to outrun them. As is their custom, wildlings move in pairs. If there is an odd number of wildlings in the village, one remains behind and is able to hide from the wights and survive. For every pair that flees a village, however, only one of them makes it to the next village to the west. Of course, as soon as the survivors arrive at the next village to join its wildlings, they must all pair up and flee to the next village to the west. Once the wights reach the westernmost village, the attack is over.

As an advisor to the Night's Watch, you are to calculate the wildling village populations so that future war plans can be formulated. You decide to do the calculations for several scenarios since the exact populations of the villages are not currently known.

Details of the input

The first line of input is a single integer, *n*, that represents the number of cases you will analyze. For each of the *n* cases, there is a single line of input containing 20 non-negative integers (each <= 1000) representing the populations of the villages. The populations are given in order with the first integer on the line representing the population of the village that is furthest west and the last integer the population of the village that is furthest east.

Details of the output

You will output a single line containing 20 integers for each case. The line begins with a label of the form "`Case i:`" (*i* = 1...*c*) followed by the integers that specify the populations of the villages after the attack in order with the westernmost village listed first and the easternmost village listed last.

Sample input

```
1
0 0 0 0 77 0 0 99 0 0 0 40 0 0 0 17 0 1 13 10
```

Sample output

```
Case 1: 5 1 0 0 1 1 0 1 1 0 0 1 0 0 1 1 1 0 0 0
```

Nested Ring Stacking

Before the children of Winterfell are old enough to play Human-White Walker, they entertain themselves with a version of the ring stacking game. In the traditional ring stacking game, rings of different sizes and colors are placed on a post with the largest ring on the bottom and the smallest on top. In the Winterfell variation of the game, there is no post and stacks of nested rings are formed. Rings are allowed to be placed inside each other (if they fit) to create nests and then nests are placed on top of each other. The challenge is to form the smallest possible stack of nest, using all rings. You can't place two small rings side-by-side inside a large ring, but you can nest as many rings inside each other as will fit together. Each ring has the same height, and is one inch thick. So the diameter of the "hole" in the ring is the diameter of the ring -2 (one inch on both sides). A ring can fit inside another ring if its diameter is ≤ the size of the hole it is trying to fit into.

Details of the input

The first line of the input will be a positive integer *n*, indicating the number of test cases. There will then follow *n* lines. Each line will start with an integer *r* ($1 \leq r \leq 100$) listing the number of rings in this test case. The line will then have *r* more entries, all separated by spaces, holding the diameters of the various rings, in inches. Each diameter will be a positive integer between 2 and 10,000.

Details of the output

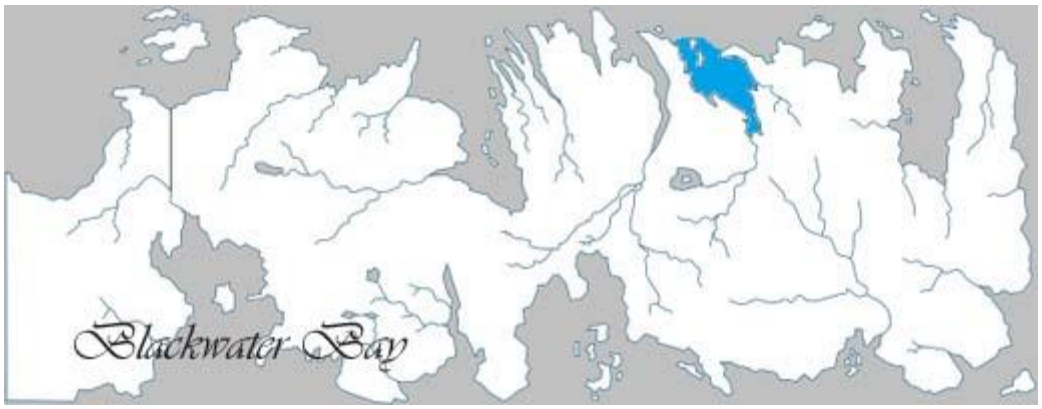
For each case, a single line of output is generated. The line is of the form "`Stack c is r rings tall.`" where *c* is the current test case (starting from 1), and *r* is the minimum height (in rings) needed to stack all of the rings.

Sample input

```
3
5 2 3 4 5 6
6 12 2 8 6 4 10
5 3 3 3 3 3
```

Sample output

```
Stack 1 is 2 rings tall.
Stack 2 is 1 rings tall.
Stack 3 is 5 rings tall.
```



Coral in Blackwater Bay

Recently, the "Mad King" Ares has been concerned over coral growth in Blackwater Bay. Apparently it was discovered that coral from the bottom of the bay is growing exponentially, and the king fears that it may impede trade routes through the Narrow Sea and affect naval war efforts in the event of a Dothraki invasion. Being an excellent scientist, you know that coral is harmless, but you're also smart enough not to argue with Ares, worthy of the title "The Mad King". You decide to fulfill Ares' request that you predict further coral growth.

While studying the coral, you figure out that coral can be modeled into "cubes". When a coral cube grows, each face of the cube that is exposed to water will grow a new cube. So if a cube sits on the bottom of the bay, which we can think of as a grid, it will grow new cubes from all of its faces, with the exception of its bottom since that sits on the floor of the bay. Once this growth process is complete, this cube will no longer grow since is completely covered by the new coral cubes. A cube that can no longer grow is considered dead.

Adding to the complexity of the situation, sediment from the Narrow Sea periodically drops into the bay. As the sediment settles to the bottom of the bay it blocks the growth of the coral it surrounds on all sides that it covers. For those cubes on an even level with the sediment, the top face remains exposed and can still grow. If more sediment drops at a later time, the sediment continues to accumulate in height evenly throughout the bay. So a deposit of depth 1 will lay on the bay's floor and cover the sides of all cubes that currently are on the bay floor as well as all of the bay floor surrounding the cubes. If the next sediment event is a deposit of depth 2, this results in the sediment being 3 blocks high (depth 3).

To start the calculations, you will be given the location of single coral cubes along the bottom of the bay. Cubes grow each time increment. Sediment deposit events, when they occur, happen at the end of the time interval. For each sediment deposit you will be told the time increment after which it occurred and its depth.

Details of the input

The first line of input will be a single integer, n, that indicates the number of test cases that will follow. For each test case there are three lines of input. The first line contains three positive integers, S, E, and T, indicating there are S initial coral blocks, E sediment events and T time steps in the simulation. ($S < 20$, $E < 10$, $T < 200$.) The 2nd line will be of the form $x_1\ y_1\ x_2\ y_2 \ldots x_S\ y_S$, where each (x_i , y_i) is the coordinates of the ith block. All values are integers and $-100 < x_i , y_i < 100$. The 3rd line is of the form $t_1\ d_1 \ldots t_E\ d_E$ indicating that at the end of time period t_i sediment of depth d_i is deposited. All are positive integers, and $t_1 < t_2 < \cdots < t_E < T$. Total sediment depth will never exceed 200.

Details of the output

For each test case, one line of output is given beginning with a label of the form "`Case i:`" (i = 1...c) followed by the total number of coral blocks, both living and dead, in the colony.

Sample input

```
3
2 1 3
1 1 3 2
1 1
1 3 4
1 1
1 2 2 1 3 1
5 3 100
-2 -2 -1 -1 0 0 1 1 1 10
3 2 15 1 27 13
```

Sample output

```
Case 1: 44
Case 2: 9
Case 3: 508087
```