Problem A –
“Bubble Gum, Bubble Gum, in the dish, how many pieces do you wish?”

(Problem appeared in the 2007/2008 Regional Competition in North America – Pacific Northwest.)

Alex and Karyn were at it again. The elementary school sisters were playing their favorite game to decide who gets to play on the computer next.

The rules of the game are quite simple. Given p people ($p > 0$), one of the p people is chosen to pick a number n ($n > p$) representing the number of pieces of bubble gum desired. Once this value is chosen, the people are iterated through, one at a time, starting at 1, from “left” to “right”, starting with the person who chose the number. Iterating is done in a circular fashion, meaning that once the person on the far right is reached, the next person in the iteration will be the person on the far left. Upon reaching n, the person at that location is the winner.

Given a list of names, followed by the name of the person choosing the number of pieces of bubble gum, followed by the number that person chose, determine who wins the game.

Input

The first value in the input file will be an integer t ($0 < t < 1000$) representing the number of test cases in the input file. Following this, on a case by case basis, will be a list of the names of the people (p), on a single line. Names will be no larger than 20 characters in length and all names are unique. There will be no more than 20 names. Each name is followed by a space, save for the last name, which is followed by a newline. On the next line is the name of the person choosing the number of pieces of bubble gum, followed by a newline. The test case is concluded with the number of pieces of gum n ($p < n < 1000$), which is also followed by a newline.

Output

For each test case, report the name of the person that won the game, followed by a newline.

Sample Input

```
3
Alex Karyn Maude
Karyn
5
Alex Karyn Maude
Alex
6
Alex Karyn Zach Becca Maude
Zach
8
```

Sample Output

```
Maude
Maude
Maude
```

Problem B – “He is Offside!”

(This problem appeared in the 2007/2008 Regional Competition in Latin America – South America.)

Hemisphere Network is the largest television network in Tumbolia, a small country located east of South America (or south of East America). The most popular sport in Tumbolia, unsurprisingly, is soccer; many games are broadcast every week in Tumbolia.

Hemisphere Network receives many requests to replay dubious plays; usually, these happen when a player is deemed to be offside by the referee. An attacking player is offside if he is nearer to his opponents' goal line than the second last opponent. A player is not offside if:

- he is level with the second last opponent or
- he is level with the last two opponents.

Through the use of computer graphics technology, Hemisphere Network can take an image of the field and determine the distances of the players to the defending team's goal line, but they still need a program that, given these distances, decides whether a player is offside.

Input

The input file contains several test cases. The first line of each test case contains two integers A and D separated by a single space indicating, respectively, the number of attacking and defending players involved in the play ($2 \le A, D \le 11$). The next line contains A integers B_i separated by single spaces, indicating the distances of the attacking players to the goal line ($1 \le B_i \le 10^4$). The next line contains D integers C_j separated by single spaces, indicating the distances of the defending players to the goal line ($1 \le C_j \le 10^4$). The end of input is indicated by A=D=0.

Output

For each test case in the input print a line containing a single character: `Y` (uppercase) if there is an attacking player offside, and `N` (uppercase) otherwise.

Sample Input

```
2 3
500 700
700 500 500
2 2
200 400
200 1000
3 4
530 510 490
480 470 50 310
0 0
```

Sample Output

```
N
Y
N
```

Problem C – “Candy Sharing Game”

(This problem was originally taken from the 2003/2004 Regional Competition in North America – Greater New York.)

A number of students sit in a circle facing their teacher in the center. Each student initially has an even number of pieces of candy. When the teacher blows a whistle, each student simultaneously gives half of his or her candy to the neighbor on the right. Any student, who ends up with an odd number of pieces of candy, is given another piece by the teacher. The game ends when all students have the same number of pieces of candy.

Write a program which determines the number of times the teacher blows the whistle and the final number of pieces of candy for each student from the amount of candy each child starts with.

Input:

The input may describe more than one game. For each game, the input begins with the number N of students, followed by N (even) candy counts for the children counter-clockwise around the circle. The input ends with a student count of 0. Each input number is on a line by itself.

Output:

For each game, output the number of rounds of the game followed by the amount of candy each child ends up with, both on one line.

Notes:

The game ends in a finite number of steps because:

1. The maximum candy count can never increase.
2. The minimum candy count can never decrease.
3. No one with more than the minimum amount will ever decrease to the minimum.
4. If the maximum and minimum candy counts are not the same, at least one student with the minimum amount must have their count increase.

Sample Input:

```
6
36
2
2
2
2
2
11
22
20
18
16
14
12
10
8
6
4
2
4
2
4
6
8
0
```

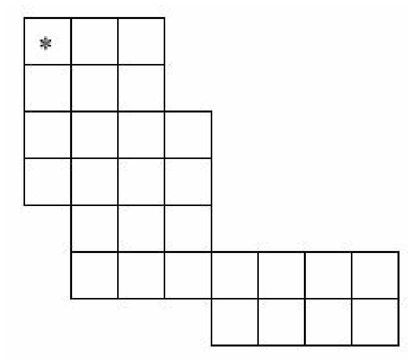
Sample Output:

```
15 14
17 22
4 8
```

Problem D – “Making Jumps”

(This problem has been modified from one that appeared in the 2003/2004 Regional Competition in North America – North Central.)

A knight is a piece used in the game of chess. The chessboard itself is square array of cells. Each time a knight moves, its resulting position is two rows and one column, or two columns and one row away from its starting position. Thus a knight starting on row *r*, column *c* - which we'll denote as (*r*, *c*) - can move to any of the squares (*r* - 2, *c* - 1), (*r* - 2, *c* + 1), (*r* - 1, *c* - 2), (*r* - 1, *c* + 2), (*r* + 1, *c* - 2), (*r* + 1, *c* + 2), (*r* + 2, *c* - 1), or (*r* + 2, *c* + 1). Of course, the knight may not move to any square that is not on the board.



Suppose the chessboard is not square, but instead has rows with variable numbers of columns, and with each row offset zero or more columns to the right. The figure to the left illustrates one possible configuration. How many of the squares in such a modified chessboard can a knight, starting in the upper left square (marked with an asterisk), not reach in any number of moves without resting in any square more than once?

If necessary, the knight is permitted to pass over regions that are outside the borders of the modified chessboard, but as usual, it can only move to squares that are within the borders of the board.

Input

There will be multiple cases to consider. The input for each case begins with an integer *n*, between 1 and 10, that specifies the number of rows in the modified chessboard. Following n there will be *n* pairs of integers, with the *i*-th pair corresponding to the *i*-th row of the chessboard. The first integer of each pair indicates the number of squares skipped at the beginning of the row. The second integer indicates the number of squares in the row (which will always be at least 1). The last case will be followed by the integer 0.

For example, input for the case illustrated by the chessboard shown above would be:

```
7 0 3 0 3 0 4 0 4 1 3 1 7 4 4
```

The maximum dimensions of the board will be 10 rows and 10 columns. That is, any modified chessboard specified by the input will fit completely on a 10 row, 10 column board.

Output

For each input case, display the case number (1, 2,...), and the number of squares that the knight can not reach. Display the results in the format shown in the examples on the next page.

Sample Input

```
7 0 3 0 3 0 4 0 4 1 3 1 7 4 4
2 0 3 0 3
3 0 3 0 3 0 3
2 0 1 2 1
0
```

Sample Output

```
Case 1, 0 squares can not be reached.
Case 2, 4 squares can not be reached.
Case 3, 1 square can not be reached.
Case 4, 0 squares can not be reached.
```

Problem E – "Making Pals"

(This problem appeared in the 2003/2004 Regional Competition in North America – North Central.)

A palindrome is a sequence that is the same when read forward or backward. For example, "pop" is a palindrome, as are "Poor Dan is in a droop" (ignoring spaces and case), and "12321".

In this problem, you are to find the "cheapest" way to transform a sequence of decimal digits into a palindrome. There are only two types of modifications you may make to the sequence, but each of these may be repeated as many times as necessary. You may delete a digit from either end of the sequence, or you may add a digit to either end of the sequence. Each of these operations incurs a "cost" of 1. For each input sequence, determine the smallest cost of transforming the sequence into a palindrome, and the length of the resulting palindrome. If two palindromes can be produced with the same cost, the length of the longer palindrome (the one with more digits) is to be reported.

For example, suppose the initial sequence was "911". This can be transformed into a palindrome by deleting the leading "9" (yielding "11") or by adding an additional "9" to the right end of the sequence (yielding "9119"). Since both of these transformations have a cost of 1, and the second transformation yields a longer palindrome, it is this one which would be reported as your result.

Note that the particular palindrome produced by the cheapest sequence of transformations is not necessarily unique, but since you are not required to report the resulting palindrome, any of these will suffice.

Input

There will be multiple cases to consider. Each case has a single line of input that contains one or more decimal digits followed by the end of line. The maximum number of digits in a sequence will be 6. The last case is followed by an empty line (that is, only an end of line).

Output

For each input case, display the case number (1, 2,...), the input sequence, the cost of the cheapest transformation, and the length of the resulting palindrome. Your output should follow the format shown in the examples on the next page.

Sample Input

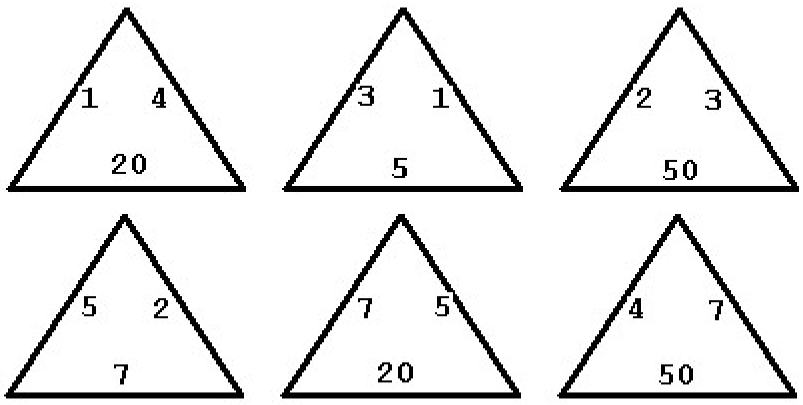
```
911
9118
11234
          <-- This line is blank
```

Sample Output

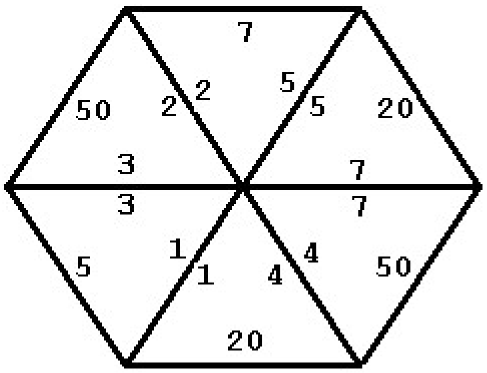
```
Case 1, sequence = 911, cost = 1, length = 4
Case 2, sequence = 9118, cost = 2, length = 4
Case 3, sequence = 11234, cost = 3, length = 8
```

Problem F – “The Triangle Game”

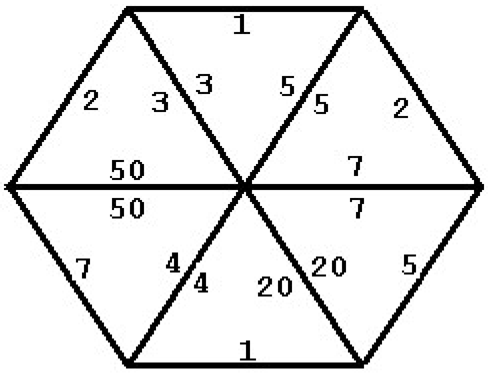
(This problem appeared in the 2000/2001 Regional Competition in North America – Mid Central.)



In the triangle game you start off with six triangles numbered on each edge, as in the example above. You can slide and rotate the triangles so they form a hexagon, but the hexagon is only legal if edges common to two triangles have the same number on them. You may not flip any triangle over. Two legal hexagons formed from the six triangles are illustrated below.



20+5+50+7+20+50 = 152



1+7+2+1+2+5 = 18

The score for a legal hexagon is the sum of the numbers on the outside six edges.

Your problem is to find the highest score that can be achieved with any six particular triangles.

Input

The input file will contain one or more data sets. Each data set is a sequence of six lines with three integers from 1 to 100 separated by blanks on each line. Each line contains the numbers on the triangles in clockwise order. Data sets are separated by a line containing only an asterisk. The last data set is followed by a line containing only a dollar sign.

Output

For each input data set, the output is a line containing only the word "none" if there are no legal hexagons or the highest score if there is a legal hexagon.

Sample Input

```
1 4 20
3 1 5
50 2 3
5 2 7
7 5 20
4 7 50
*
10 1 20
20 2 30
30 3 40
40 4 50
50 5 60
60 6 10
*
10 1 20
20 2 30
30 3 40
40 4 50
50 5 60
10 6 60
$
```

Sample Output

```
152
21
none
```