Problem A: Misspelling

Misspelling is an art form that students seem to excel at. Write a program that removes the *M*th character from an input string.

Input
The first line of input contains a single integer ***N***, (1 ≤ ***N*** ≤ 1000) which is the number of datasets that
follow.
Each dataset consists of a single line of input containing ***M***, a space, and a single word made up of uppercase letters only. ***M*** will be less than or equal to the length of the word. The length of theword is guaranteed to be less than or equal to 80.

Output
For each dataset, you should generate one line of output with the following values: The dataset number as a decimal integer (start counting at one), a space, and the misspelled word. The misspelled word is the input word with the indicated character deleted.

Sample Input	Sample Output

`4`
`4 MISSPELL`
`1 PROGRAMMING`
`7 CONTEST`
`3 BALLOON`
	`1 MISPELL`
	`2 ROGRAMMING`
	`3 CONTES`
	`4 BALOON`

Problem B: 01000001

Adding binary numbers is a very simple task, and very similar to the longhand addition of decimal numbers. As with decimal numbers, you start by adding the bits (digits) one column at a time, from right to left. Unlike decimal addition, there is little to memorize in the way of rules for the addition of binary bits:

```
0 + 0 = 0
1 + 0 = 1
0 + 1 = 1
1 + 1 = 10
1 + 1 + 1 = 11
```

Just as with decimal addition, when the sum in one column is a two-bit (two-digit) number, the least significant figure is written as part of the total sum and the most significant figure is "carried" to the next left column. Consider the following examples:

```
                      11  1 <-- Carry bits --> 1   11
  1001101               1001001                   1000111
+ 0010010             + 0011001                 + 1010110
---------             ---------                 ---------
  1011111               1100010                  10011101
```

The addition problem on the left did not require any bits to be carried, since the sum of bits in each column was either 1 or 0, not 10 or 11. In the other two problems, there definitely were bits to be carried, but the process of addition is still quite simple.

Input

The first line of input contains an integer ***N***, (`1 <= N <= 1000`), which is the number of binary addition problems that follow. Each problem appears on a single line containing two binary values separated by a single space character. The maximum length of each binary value is 80 bits (binary digits). Note: The maximum length result could be 81 bits (binary digits).

Output

For each binary addition problem, print the problem number, a space, and the binary result of the addition. Extra leading zeroes must be omitted.

Sample Input	Sample Output

3
1001101 10010
1001001 11001
1000111 1010110
	1 1011111
	2 1100010
	3 10011101

Problem C: Digital Roots

Background

The *digital root* of a positive integer is found by summing the digits of the integer. If the resulting value is a single digit then that digit is the digital root. If the resulting value contains two or more digits, those digits are summed and the process is repeated. This is continued as long as necessary to obtain a single digit.

For example, consider the positive integer 24. Adding the 2 and the 4 yields a value of 6. Since 6 is a single digit, 6 is the digital root of 24. Now consider the positive integer 39. Adding the 3 and the 9 yields 12. Since 12 is not a single digit, the process must be repeated. Adding the 1 and the 2 yeilds 3, a single digit and also the digital root of 39.

Input

The input file will contain a list of positive integers($<10^{80}$), one per line. The end of the input will be indicated by an integer value of zero.

Output

For each integer in the input, output its digital root on a separate line of the output.

Sample Input	Sample Output
24 39 0	6 3

Problem D: Number

Find the smallest natural number that is not presented in the input.

Input
The first line of input contains integer number N (0< N < 10000). Next line contains N-1 different integer numbers from the range 1..N.

Output
Output one number 1..N that is not presented in the input.

Sample Input	Sample Output
4 1 2 3	4
5 2 1 4 5	3

Problem E: Recursively Palindromic Partitions

A *partition* of a positive integer ***N*** is a sequence of integers which sum to ***N***, usually written with plus signs between the numbers of the partition.
For example

`15 = 1+2+3+4+5 = 1+2+1+7+1+2+1`

A partition is *palindromic* if it reads the same forward and backward. The first partition in the example is *not* palindromic while the second is. If a partition containing ***m*** integers is palindromic, its left half is the first **`floor(m/2)`** integers and its right half is the last **`floor(m/2)`** integers (which must be the reverse of the left half. (**`floor(x)`** is the greatest integer less than or equal to ***x***.)

A partition is *recursively palindromic* if it is palindromic and its left half is recursively palindromic or empty. Note that every integer has at least two recursively palindromic partitions one consisting of all ones and a second consisting of the integer itself. The second example above is also recursively palindromic.
For example, the recursively palindromic partitions of 7 are:

`7, 1+5+1, 2+3+2, 1+1+3+1+1, 3+1+3, 1+1+1+1+1+1+1`

Write a program which takes as input an integer ***N*** and outputs the number of recursively palindromic partitions of ***N***.

Input

The first line of input contains a single integer ***T***, ($1 \le T \le 1000$) which is the number of data sets that follow. Each data set consists of a single line of input containing a single positive integer **N** ($1 \le N \le 1000$) for which the number of recursively palindromic partitions is to be found.

Output

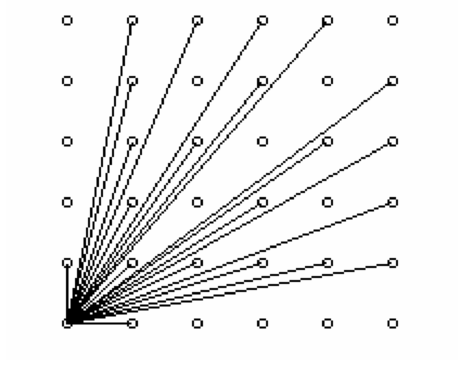
For each data set, you should generate one line of output with the following values: The data set number as a decimal integer (start counting at one), a space and the number of recursively palindromic partitions of the input value.

Sample Input	Sample Output

3
4
7
20
	1 4
	2 6
	3 60

Problem F: Visible Lattice Points

A lattice point (***x***, ***y***) in the first quadrant (***x*** and ***y*** are integers greater than or equal to 0), other than the origin, is *visible* from the origin if the line from (0, 0) to (***x***, ***y***) does not pass through any other lattice point. For example, the point (4, 2) is not visible since the line from the origin passes through (2, 1). The figure below shows the points (***x***, ***y***) with 0 ≤ ***x***, ***y*** ≤ 5 with lines from the origin to the visible points.



Write a program which, given a value for the size, **N**, computes the number of visible points (***x***,***y***) with 0 ≤ ***x***, ***y*** ≤ **N**.

Input

The first line of input contains a single integer **C**, (1 ≤ **C** ≤ 1000) which is the number of datasets that follow. Each dataset consists of a single line of input containing a single integer **N**, (1 ≤ **N** ≤ 1000), which is the size.

Output

For each dataset, there is to be one line of output consisting of: the dataset number starting at 1, a single space, the size, a single space and the number of visible points for that size.

Sample Input	Sample Output

4
2
4
5
231
	1 2 5
	2 4 13
	3 5 21
	4 231 32549