1. *Collatz Conjecture*. The conjecture is summarized as follows:

> Take any positive integer ***n***. If ***n*** is 1, stop. If ***n*** is even, divide it by 2 using integer division. If ***n*** is odd, multiply it by 3 and add 1. Repeat this process.

You are to write a Java program that given a starting number prints the number of steps to reach 1.

Input: the first line gives an integer ***k*** which is the number of test cases and is followed by ***k*** lines of inputs, each of which gives the ***n*** for a test case (as defined above).

- Example:

```
5
1
3
4
23
113383
```

Output: ***k*** lines of output, each of which is a single integer (for each test case) representing the number of steps (as defined above).

- Example (the output for the above inputs):

```
0
7
2
15
247
```

(***Hint***: use data type *long* rather than *int* for the calculation.)

2. *RooksOnChessboard*. This game is to place rooks on a chessboard (an 8x8 board with positions indexed from (1, 1) to (8, 8)) in a way that no rook can threaten another rook. You are to write a Java program to determine whether any rooks are threatened (note: rooks move along rows and columns: this means that two rooks may not be on the same row or column).

Input: the first line gives an integer ***k*** which is the number of test cases (i.e., number of chessboards) and is followed by ***k*** lines of inputs, each of which (for a test case) begins with the number of rooks followed by the column and row positions of each rook.

- Example:

```
3
3 1 1 2 6 8 8
2 2 3 1 3
3 6 2 8 3 7 4
```

Output: ***k*** lines of output, each of which outputs "Safe" or "Threatened" on a single line.

- Example (the output for the above inputs):

```
Safe
Threatened
Safe
```

3. *Gas Station*. People usually drive far away from their neighborhood gas station to get the best price. You are to write a Java program that, given a list of gas prices and distances from a person's home (for a set of gas stations), determines which gas station will cost the least. (The cost of buying gas from a station includes the price of the gas itself, plus the cost of the gas used to drive to and from the gas station, plus the cost of wear and tear on the car driving to and from the station.)

Input: the first line gives an integer ***k*** which is the number of test cases and is followed by ***k*** data sets for the test cases. Each data set is defined as follows:

- The first line in each data set consists of 4 numbers: ***X G M W***:
  - ***X***: an integer, the number of gas stations in the data set (***X*** >=1).
  - ***G***: a floating point number, the number of gallons the drive will buy.
  - ***M***: a floating point number, the miles per gallon the driver's car uses.
  - ***W***: a floating point number, the cost in wear and tear per mile for the car.
- The next ***X*** lines of data set are gas stations. Each line consists of two floating point numbers ***D P***:
  - ***D***: the distance in miles from the driver's home to the gas station.
  - ***P***: the price per gallon for gas at that station.

- Example:

```
3
3 15.7 17.2 0.27
3.6 3.75
40.5 2.99
10.0 3.05
1 8.8 25.0 0.22
0.8 3.25
1 100.0 20.0 0.25
100.0 4.25
```

Output: ***k*** lines of output, each of which consists of two numbers: the number of the gas station with the cheapest total cost for the desired gas (the first gas station in a data set is number 1, the second is number 2, and so forth), and the lowest total cost for the desired amount of gas from the station rounded to the nearest cent.

- Example (the output for the above inputs):

```
3 56.83
1 29.16
1 517.50
```

4. *Look-and-Say Sequence*. The *Look-and-Say* sequence is a recursively-defined sequence of numbers (or strings). The definition reads like:

- Take a string (a number or a string; this is considered a seed at the beginning).
- Look at the string, visually grouping consecutive runs of the same digit/char.
- Say the number, from left to right, group by group; as how many of that digit/char there are—followed by the digit/char grouped.
- (This becomes the next string of the sequence.)

For example, the *Look-and-Say* sequence defined with seed "1" is produced in the following steps:

(1) Starting with 1, you have *one* 1 which produces 11;
(2) Starting with 11, with have *two* 1's (i.e., 21);
(3) Starting with 21, you have *one* 2, then *one* 1, i.e., (12)(11) which becomes 1211;
(4) Starting with 1211, you will have 111221…

In this way, you have the following sequence for a starting number, *seed*, "1": 11, 21, 1211, 111221, 312211, 13112221…

More examples of *Look_and_Say* sequences are given below:

- Seed "2": 12, 1112, 3112, 132112, 1113122112, …
- Seed "555": 35, 1315, 11131115, 31133115, 1321232115, …
- Seed "abb": 1a2b, 111a121b, 311a1112111b, 13211a3112311b, 111312211a13211213211b,…

You are to write a Java program that outputs the ***n***-th item of a sequence given a particular seed ***s***. You are not to produce all the items for the sequences, but only one item (the ***n***-th one) per sequence.

Input: the first line gives an integer ***k*** which is the number of test cases and is followed by ***k*** data sets for the test cases. Each data set consists of two lines for a test case: the first line is ***s*** (i.e., the seed) and the second line is ***n*** (the ***n***-th item in the sequence to be outputted).

- Example:

```
3
1
1
555
3
abb
2
```

Output: ***k*** lines of output, each of which is a single sequence item defined above.

- Example (the output for the above inputs):

```
11
11131115
111a121b
```

5. *NoConsecutive00*. Write a Java program that calculates the number of binary sequences (i.e., binary bit patterns) of length ***n*** that have no consecutive 0's. For example, for ***n*** = 3, we have five such sequences: 010, 011, 101, 110, and 111.

Input: the first line gives an integer ***k*** which is the number of test cases and is followed by ***k*** lines of inputs, each of which gives the value of ***n*** (as defined above, and ***n*** > 0) for a test case.

- Example:

```
6
3
5
10
18
21
87
```

Output: ***k*** lines of output, each of which is a single integer number (for each test case) representing the number of binary sequences (as defined above).

- Example (the output for the above inputs):

```
5
13
144
6765
28657
1779979416004714189
```