107 - The Cat in the Hat

Time limit: 3.000 seconds

Background

(An homage to Theodore Seuss Geisel)

The Cat in the Hat is a nasty creature,
But the striped hat he is wearing has a rather nifty feature.

With one flick of his wrist he pops his top off.

Do you know what's inside that Cat's hat?
A bunch of small cats, each with its own striped hat.

Each little cat does the same as line three,
All except the littlest ones, who just say ``Why me?"

Because the littlest cats have to clean all the grime,
And they're tired of doing it time after time!

The Problem

A clever cat walks into a messy room which he needs to clean. Instead of doing the work alone, it decides to have its helper cats do the work. It keeps its (smaller) helper cats inside its hat. Each helper cat also has helper cats in its own hat, and so on. Eventually, the cats reach a smallest size. These smallest cats have no additional cats in their hats. These unfortunate smallest cats have to do the cleaning.

The number of cats inside each (non-smallest) cat's hat is a constant, N. The height of these cats-in-a-hat is $\frac{1}{1+N}$ times the height of the cat whose hat they are in.

The smallest cats are of height one;
these are the cats that get the work done.
All heights are positive integers.

Given the height of the initial cat and the number of worker cats (of height one), find the number of cats that are not doing any work (cats of height greater than one) and also determine the sum of all the cats' heights (the height of a stack of all cats standing one on top of another).

The Input

The input consists of a sequence of cat-in-hat specifications. Each specification is a single line consisting of two positive integers, separated by white space. The first integer is the height of the initial cat, and the second integer is the number of worker cats.

A pair of 0's on a line indicates the end of input.

The Output

For each input line (cat-in-hat specification), print the number of cats that are not working, followed by a space, followed by the height of the stack of cats. There should be one output line for each input line other than the ``0 0'' that terminates input.

Sample Input

```
216 125
5764801 1679616
0 0
```

Sample Output

```
31 671
335923 30275911
```

10042 - Smith Numbers

Time limit: 3.000 seconds

Background

While skimming his phone directory in 1982, Albert Wilansky, a mathematician of Lehigh University , noticed that the telephone number of his brother-in-law H. Smith had the following peculiar property: The sum of the digits of that number was equal to the sum of the digits of the prime factors of that number. Got it? Smith's telephone number was 493-7775. This number can be written as the product of its prime factors in the following way:

$$4937775 = 3 \cdot 5 \cdot 5 \cdot 65837$$

The sum of all digits of the telephone number is 4+9+3+7+7+7+5=42, and the sum of the digits of its prime factors is equally 3+5+5+6+5+8+3+7=42. Wilansky was so amazed by his discovery that he named this type of numbers after his brother-in-law: Smith numbers.

As this observation is also true for every prime number, Wilansky decided later that a (simple and unsophisticated) prime number is not worth being a Smith number and he excluded them from the definition.

Problem

Wilansky published an article about Smith numbers in the *Two Year College Mathematics Journal* and was able to present a whole collection of different Smith numbers: For example, 9985 is a Smith number and so is 6036. However, Wilansky was not able to give a Smith number which was larger than the telephone number of his brother-in-law. It is your task to find Smith numbers which are larger than 4937775.

Input

The input consists of several test cases, the number of which you are given in the first line of the input.

Each test case consists of one line containing a single positive integer smaller than 10^9.

Output

For every input value *n*, you are to compute the smallest Smith number which is larger than *n* and print each number on a single line. You can assume that such a number exists.

Sample Input

```
1
4937774
```

Sample Output

```
4937775
```

10810 - Ultra-QuickSort

Time limit: 3.000 seconds
In this problem, you have to analyze a particular sorting algorithm. The algorithm processes a sequence of *n* distinct integers by swapping two adjacent sequence elements until the sequence is sorted in ascending order. For the input sequence
`9 1 0 5 4,`
Ultra-QuickSort produces the output
`0 1 4 5 9.`
Your task is to determine how many swap operations Ultra-QuickSort needs to perform in order to sort a given input sequence.

The input contains several test cases. Every test case begins with a line that contains a single integer $n < 500,000$ -- the length of the input sequence. Each of the the following *n* lines contains a single integer $0 \le a[i] \le 999,999,999$, the *i*-th input sequence element. Input is terminated by a sequence of length $n = 0$. This sequence must not be processed.

For every input sequence, your program prints a single line containing an integer number *op*, the minimum number of swap operations necessary to sort the given input sequence.

Sample Input

```
5
9
1
0
5
4
3
1
2
3
0
```

Output for Sample Input

```
6
0
```

10044 - Erdos Numbers

Time limit: 3.000 seconds

Background

The Hungarian Paul Erdös (1913-1996, speak as ``Ar-dish'') not only was one of the strangest mathematicians of the 20th century, he was also one of the most famous. He kept on publishing widely circulated papers up to a very high age and every mathematician having the honor of being a co-author to Erdös is well respected.

Not everybody got the chance to co-author a paper with Erdös, so many people were content if they managed to publish a paper with somebody who had published a scientific paper with Erdös. This gave rise to the so-called *Erdös numbers*. An author who has jointly published with Erdös had Erdös number 1. An author who had not published with Erdös but with somebody with Erdös number 1 obtained Erdös number 2, and so on.

Problem

Today, nearly everybody wants to know which Erdös number he or she has. Your task is to write a program which computes Erdös numbers for a given set of scientists.

Input

The first line of the input contains the number of scenarios.

The input for each scenario consists of a paper database and a list of names. It begins with the line

$P\ N$

where P and N are natural numbers. Following this line are P lines containing descriptions of papers (this is the paper database). A paper appears on a line by itself and is specified in the following way:

```
Smith, M.N., Martin, G., Erdos, P.: Newtonian forms of prime factors
matrices
```

Note that umlauts like `ö' are simply written as `o'. After the P papers follow N lines with names. Such a name line has the following format:

```
Martin, G.
```

Output

For every scenario you are to print a line containing a string ``Scenario i'' (where i is the number of the scenario) and the author names together with their Erdös number of all authors in the list of names. The authors should appear in the same order as they appear in the list of names. The Erdös number is based on the papers in the paper database of this scenario. Authors which do not have any relation to Erdös via the papers in the database have Erdös number ``infinity''.

Sample Input

```
1
4 3
Smith, M.N., Martin, G., Erdos, P.: Newtonian forms of prime factor
matrices
```

```
Erdos, P., Reisig, W.: Stuttering in petri nets
Smith, M.N., Chen, X.: First oder derivates in structured programming
Jablonski, T., Hsueh, Z.: Selfstabilizing data structures
Smith, M.N.
Hsueh, Z.
Chen, X.
```

Sample Output

```
Scenario 1
Smith, M.N. 1
Hsueh, Z. infinity
Chen, X. 2
```

948 - Fibonaccimal Base

Time limit: 3.000 seconds

The well known Fibonacci sequence is obtained by starting with 0 and 1 and then adding the two last numbers to get the next one. For example the third number in the sequence is 1 (1=1+0), the forth is 2 (2=1+1), the fifth is 3 (3=2+1) and so on.

i	0	1	2	3	4	5	6	7	8	9
Fib(i)	0	1	1	2	3	5	8	13	21	34

Figure 1 - The first numbers in the Fibonacci sequence

The sequence appears on many things in our life, in nature, and has a great significance. Among other things, do you know that all positive integer numbers can be represented as a sum of numbers in the Fibonacci sequence? More than that, all positive integers can be represented as a sum of a set of Fibonacci numbers, that is, numbers from the sequence, without repetition. For example: *13* can be the sum of the sets *{13}*, *{5,8}* or *{2,3,8}* and *17* is represented by *{1,3,13}* or *{1,3,5,8}*. Since all numbers have this property (do you want to try to prove this for yourself?) this set could be a nice way to use as a "base" to represent the number. But, as we have seen, some numbers have more than one set whose sum is the number. How can we solve that? Simple! If we add the constraint that the sets cannot have two consecutive Fibonacci numbers, than we have a unique representation for each number! This restriction is because the sum of any two consecutive Fibonacci numbers is just the following Fibonacci number.

Now that we know all this we can prepare a nice way to represent any positive integer. We will use a binary sequence (just zeros and ones) to do that. For example, *17 = 1 + 3 + 13* (remember that no two consecutive Fibonacci numbers can be used). Let's write a zero for each Fibonacci number that is not used and one for each one that is used, starting at the right. Then, *17 = 100101*. See figure 2 for a detailed explanation. In this representation we should not have zeros at the left, this is, we should only write starting with the first one. In order for you to understand better, note that in this scheme, not using two consecutive Fibonacci numbers means that the binary sequence will not have two consecutive ones. When we use this representation for a number we say that we are using the Fibonaccimal base, and we write it like *17 = 100101 (fib)*.

17 =	1	0	0	1	0	1
13+3+1 =	**13**	8	5	**3**	2	**1**

Figure 2 - Explaining the representation of 17 in Fibonaccimal base

The Problem

Given a set of numbers in decimal base, your task is to write them in the Fibonaccimal base.

Input

The first line of input contains a single number **N**, representing the quantity of numbers that follow ($1 \leq N \leq 500$).

Than follow exactly ***N*** lines, each one containing a single positive integer smaller than 100 000 000. These numbers can come in any order.

Output

You should output a single line for each of the ***N*** integers in the input, with the format `"DEC_BASE = FIB_BASE (fib)"`. `DEC_BASE` is the original number in decimal base and `FIB_BASE` is its representation in Fibonaccimal base. See the sample output for an example.

Example Input

```
10
1
2
3
4
5
6
7
8
9
10
```

Example Output

```
1 = 1 (fib)
2 = 10 (fib)
3 = 100 (fib)
4 = 101 (fib)
5 = 1000 (fib)
6 = 1001 (fib)
7 = 1010 (fib)
8 = 10000 (fib)
9 = 10001 (fib)
10 = 10010 (fib)
```

10419 - Sum-up the Primes

Time limit: 4.000 seconds

We all know from Goldbach's conjecture that any even number greater than 2 can be expressed as a summation of two primes. Some odd numbers can also be expressed as summation of two primes. In this problem you will have to express a number as a summation of arbitrary number of primes less than 300. The conditions in detail are as follows:

1) You have to express a number N (N<=1000) as a summation of t(t<=14) primes.

2) Among the t primes any single odd primes can be present maximum two times. 2 can be present only once. For example, (5+5+3+3) is valid, but (3+3+3+7) or (2+2+3) is invalid according to this particular rule.

3) All the prime numbers used must be less than 300.

4) If there is more than one solution print the lexicographically smallest one.

5) If there is no such expression of primes print the string "No Solution."

Input

The input file contains less than 9340 lines of input. Each line contains two numbers N (0<N<=1000) and t (0<t<=14). The meaning of N and t are described in the problem statement. Input is terminated by a line where N=0 and t=0. This line should not be processed.

Output

For each line of input produce a block of 2 lines. The first line of such a block contains the output serial as shown in the sample output. Next line contains the lexicographically smallest expression that sums up to N. There is no space between the operators and operands. If N cannot be expressed as a summation of t primes output "No Solution."

Sample Input

```
20 10

100 4

10 2

0 0
```

Sample Output

```
CASE 1:

No Solution.
```

```
CASE 2:

11+11+17+61

CASE 3:

3+7
```

10056 - What is the Probability ?

Time limit: 3.000 seconds

Probability has always been an integrated part of computer algorithms. Where the deterministic algorithms have failed to solve a problem in short time, probabilistic algorithms have come to the rescue. In this problem we are not dealing with any probabilistic algorithm. We will just try to determine the winning probability of a certain player.

A game is played by throwing a dice like thing (it should not be assumed that it has six sides like an ordinary dice). If a certain event occurs when a player throws the dice (such as getting a 3, getting green side on top or whatever) he is declared the winner. There can be N such player. So the first player will throw the dice, then the second and at last the N th player and again the first player and so on. When a player gets the desired event he or she is declared winner and playing stops. You will have to determine the winning probability of one (The I th) of these players.

Input

Input will contain an integer S (S<=1000) at first, which indicates how many sets of inputs are there. The next S lines will contain S sets of inputs. Each line contain an integer N (N<=1000) which denotes the number players, a floating point number p which indicates the probability of the happening of a successful event in a single throw (If success means getting 3 then p is the probability of getting 3 in a single throw. For a normal dice the probability of getting 3 is 1/6), and I (I<=N) the serial of the player whose winning probability is to be determined (Serial no varies from 1 to N). You can assume that no invalid probability (p) value will be given as input.

Output

For each set of input, output in a single line the probability of the I th player to win. The output floating point number will always have four digits after the decimal point as shown in the sample output.

Sample Input:

```
2
2 0.166666 1
2 0.166666 2
```

Sample Output:

```
0.5455
0.4545
```