Topic : Number Theory **25/07/2010**

575 – Skew Binary

Skew Binary

When a number is expressed in decimal, the *k*-th digit represents a multiple of 10^k. (Digits are numbered from right to left, where the least significant digit is number 0.) For example,

$$81307_{10} = 8\times10^4+1\times10^3+3\times10^2+0\times10^1+7\times10^0 = 80000+1000+300+0+7 = 81307.$$

When a number is expressed in binary, the *k*-th digit represents a multiple of 2^k. For example,

$$10011_2 = 1\times2^4+0\times2^3+0\times2^2+1\times2^1+1\times2^0 = 16+0+0+2+1 = 19.$$

In **skew binary**, the *k*-th digit represents a multiple of 2^{k+1} - 1. The only possible digits are 0 and 1, except that the least-significant nonzero digit can be a 2. For example,

$$10120_{skew} = 1\times(2^5-1)+0\times(2^4-1)+1\times(2^3-1)+2\times(2^2-1)+0\times(2^1-1) = 31+0+7+6+0 = 44.$$

The first 10 numbers in skew binary are 0, 1, 2, 10, 11, 12, 20, 100, 101, and 102. (Skew binary is useful in some applications because it is possible to add 1 with at most one carry. However, this has nothing to do with the current problem.)

Input

The input file contains one or more lines, each of which contains an integer *n*. If *n* = 0 it signals the end of the input, and otherwise *n* is a nonnegative integer in skew binary.

Output

For each number, output the decimal equivalent. The decimal value of *n* will be at most 2^{31} - 1 = 2147483647.

Sample Input

```
10120
2000000000000000000000000000000
10
1000000000000000000000000000000
11
100
11111000001110000101101102000
0
```

Sample Output

```
44
2147483646
3
2147483647
4
7
1041110737
```

350 – Pseudo – Random Numbers

Pseudo-Random Numbers

Computers normally cannot generate really random numbers, but frequently are used to generate sequences of pseudo-random numbers. These are generated by some algorithm, but appear for all practical purposes to be really random. Random numbers are used in many applications, including simulation.

A common pseudo-random number generation technique is called the linear congruential method. If the last pseudo-random number generated was L, then the next number is generated by evaluating $(Z \times L + I) \bmod M$, where Z is a constant multiplier, I is a constant increment, and M is a constant modulus. For example, suppose Z is 7, I is 5, and M is 12. If the first random number (usually called the *seed*) is 4, then we can determine the next few pseudo-random numbers are follows:

Last Random Number, L	$(Z \times L + I)$	Next Random Number, $(Z \times L + I) \bmod M$
4	33	9
9	68	8
8	61	1
1	12	0
0	5	5
5	40	4

As you can see, the sequence of pseudo-random numbers generated by this technique repeats after six numbers. It should be clear that the longest sequence that can be generated using this technique is limited by the modulus, M.

In this problem you will be given sets of values for Z, I, M, and the seed, L. Each of these will have no more than four digits. For each such set of values you are to determine the length of the cycle of pseudo-random numbers that will be generated. But be careful: the cycle might not begin with the seed!

Input

Each input line will contain four integer values, in order, for Z, I, M, and L. The last line will contain four zeroes, and marks the end of the input data. L will be less than M.

Output

For each input line, display the case number (they are sequentially numbered, starting with 1) and the length of the sequence of pseudo-random numbers before the sequence is repeated.

Sample Input

```
7 5 12 4
5173 3849 3279 1511
9111 5309 6000 1234
1079 2136 9999 1237
0 0 0 0
```

Sample Output

```
Case 1: 6
Case 2: 546
Case 3: 500
Case 4: 220
```

10879 – Code Refactoring

Problem B
Code Refactoring
Time Limit: 2 seconds

"Harry, my dream is a code waiting to be broken. Break the code, solve the crime."
Agent Cooper

Several algorithms in modern cryptography are based on the fact that factoring large numbers is difficult. Alicia and Bobby know this, so they have decided to design their own encryption scheme based on factoring. Their algorithm depends on a secret code, **K**, that Alicia sends to Bobby before sending him an encrypted message. After listening carefully to Alicia's description, Yvette says, "But if I can intercept **K** and factor it into two positive integers, **A** and **B**, I would break your encryption scheme! And the **K** values you use are at most 10,000,000. Hey, this is so easy; I can even factor it twice, into two different pairs of integers!"

Input
The first line of input gives the number of cases, **N** (at most 25000). **N** test cases follow. Each one contains the code, **K**, on a line by itself.

Output
For each test case, output one line containing "Case #**x**: **K** = **A** * **B** = **C** * **D**", where **A**, **B**, **C** and **D** are different positive integers larger than 1. A solution will always exist.

Sample Input	Sample Output
3 120 210 10000000	Case #1: 120 = 12 * 10 = 6 * 20 Case #2: 210 = 7 * 30 = 70 * 3 Case #3: 10000000 = 10 * 1000000 = 100 * 100000

---

568 - Just the Facts

Just the Facts

The expression *N*!, read as ``*N* factorial," denotes the product of the first *N* positive integers, where *N* is nonnegative. So, for example,

N	*N*!
0	1
1	1
2	2
3	6
4	24
5	120
10	3628800

For this problem, you are to write a program that can compute the last non-zero digit of any factorial for ($0 \leq N \leq 10000$). For example, if your program is asked to compute the last nonzero digit of 5!, your program should produce ``2" because 5! = 120, and 2 is the last nonzero digit of 120.

Input

Input to the program is a series of nonnegative integers not exceeding 10000, each on its own line with no other letters, digits or spaces. For each integer *N*, you should read the value and compute the last nonzero digit of *N*!.

Output

For each integer input, the program should print exactly one line of output. Each line of output should contain the value *N*, right-justified in columns 1 through 5 with leading blanks, not leading zeroes. Columns 6 - 9 must contain `` -> "
(space hyphen greater space). Column 10 must contain the single last non-zero digit of *N*!.

Sample Input

```
1
2
26
125
3125
9999
```

Sample Output

```
    1 -> 1
    2 -> 2
   26 -> 4
  125 -> 8
 3125 -> 2
 9999 -> 8
```