6300 Signed Binary Representation of Integers

Computing $a^x \bmod n$ for large integers x and n is the most time-consuming process in most public-key cryptography systems. An algorithm to compute $a^x \bmod n$ can be described in a C-like pseudo-code as follows.

Input: positive integers a, x and n,
Output: $y = a^x \bmod n$
Method:

```
convert x into binary (x_{k-1} x_{k-2} ... x_0)_2;
y = 1;
for (i = k - 1; i >= 0; i = i - 1) {
    y = y^2 mod n;
    if (x_i == 1) y = y * a mod n;
}
print y;
```

In the above algorithm, first x is converted into k-bit binary. Then the algorithm performs k iterations. In each iteration, a square ($y^2 \bmod n$) is computed. In the i-th iteration, if $x_i = 1$, the algorithm also computes $y = y \times a \bmod n$. Therefore, the computing time depends on the number of 1's in the binary representation of x.

Let a^{-1} be the inverse of a in the group $\mathbf{Z}_n^*$. That is $a \times a^{-1} \equiv 1 \bmod n$. For example, assume that $n = 13$, then the inverse of 2 is 7, $2 \times 7 = 14 \equiv 1 (\mathrm{mod} 13)$. In this problem, you do not need to know how to compute the inverses.

Assume that a^{-1} is known, and x is represented by -1, 0, 1, instead of 0, 1, we can modify the above algorithm as follows.

Input: positive integers a, x and n,
Output: $y = a^x \bmod n$
Method:

```
convert x into signed binary (x_{k-1} x_{k-2} ... x_0)_{-1,0,1};
y = 1;
for (i = k - 1; i >= 0; i = i - 1) {
    y = y^2 mod n;
    if (x_i == 1) y = y * a mod n;
    if (x_i == -1) y = y * a^{-1} mod n;
}
print y;
```

In the above algorithm, we need to represent x by using -1, 0, 1. This is called *signed binary* representation of x. For convenience, we shall use $\overline{1}$ to denote -1. For example: $15 = (1111)_2 = (1000\overline{1})_{-1,0,1}$.

You can see that it may be more efficient to use signed binary representation in computing $a^x \bmod n$ when the inverse of a ($a^{-1} \bmod n$) is known. For $x = 15$, using binary needs 4 multiplications, while using signed binary needs only 2.

In this problem, you are going to write a program to convert a large integer into signed binary. Signed binary representation of an integer may not be unique. We need a representation with minimum number of non-zero bits to make the computation of $a^x \bmod n$ fast.

A hint of converting x into a signed binary with minimum number of non-zero bits: Make sure that no adjacent two bits are both non-zero.

Input

The input to this problem is a sequence of large integers. Each integer x is no more than 120 decimal digits, and it is written in a line. There will be no spaces in front of x, and no spaces after x. The last line of the input file contains a single '0'. You do not need to process this line.

Output

The outputs for each test case consists of two parts written in one line. The first part is an integer b, which is the number of non-zero bits of the binary representation of x. The second part is an integer s, which is the number of non-zero bits in the signed binary representation of x. Print exactly one space between these two parts.

Sample Input

```
15
2514593
113113561845
0
```

Sample Output

```
4 2
11 9
23 14
```