University of Arkansas
Department of Computer Science and Computer Engineering
2015 High School Programming Contest
Problems

Wing Ning Li Hung Nguyen Adam Higgins Paul Martin Tyler Moore

Revision Date: March 4, 2015

Please carefully read the output specification. Submitted solutions are graded by a differential program, i.e., the output produced by a submission will be compared *byte-for-byte* with the output of the judges' solutions. Thus, even though submitted solutions may produce output that is *technically* correct, it will be counted as incorrect if these specifications are not followed exactly.

For all problems, solutions receive input from standard input (*stdin*) and produce output on standard output (*stdout*).

Restricting input and output to stdin and stdout does not prevent a program from processing files. Let `Main` be the program name, `Input` be the input file name, and `Output` be the output file name. The following command at the terminal allows the program to read and write the input and output files respectively:

```
./Main < Input > Output      # for C or C++, assuming Main is the executable
java Main < Input > Output   # for Java, assuming Main is a class file
```

For all problems, an *integer* is defined as an optional minus sign followed by one digit from the set 1..9 followed by zero or more digits from the set 0..9. `3`, `-21`, and `309` are integers; `0123`, `- 4`, and `+38` are not.

A *positive integer* is an integer whose value is greater than 0. You may assume the value of a positive integer is no more than $2^{31} - 1$, or $2,147,483,647$.

A *string*'s maximum length is 16,384.

For all problems, a *line* is defined as zero or more characters followed by a newline ('\n') character. Unless otherwise directed, a line of output should have no leading or trailing whitespace. A program producing the integer 27 on one line of output should consist of the character '2' followed by the character '7' followed by the newline ('\n') character.

Input for a problem consists of multiple test cases; each test case is followed by a blank line. You must read input until the *end-of-file*. Refer to the code for the Warm Up problem for hints and ideas for handling input properly.

For each test case, your program must output the result following the specified format in each problem. Output a blank line after the output of each test case. Refer to the code for the Warm Up problem again for hints and ideas for handling output properly.

1 Comparison

When two non negative integers are compared, three outcomes are possible: the two number are equal, the first number is greater, or the second number is greater.

Write a program that compares two non negative integers and output =, >, or <.

input Each test case has only one line containing two non negative integers separated by a space. *There is always a blank line after every test case.*

output For each test case, output two lines: the first line contains =, >, or < depending on the comparison outcome *and a blank line as the second.*

Example:

Input

```
1 3

1 1

5 3

45 45

1001001100 1001001100
```

Output

```
<

=

>

=

=
```

2 Comparison generalized

The two non negative integers in the previous problem are represented as decimal numbers. For this problem, the first number is still a decimal number but the second number could be either a decimal number or a number described in English. For example, "one" for 1, "seven" for 7, "sixteen" for 16, "forty five" for 45, and "one hundred" for 100. Even though "eleven hundred" and "one thousand one hundred" both equal 1100, this problem only deals with the "one thousand and one hundred" version.

Write a program that compares two non negative integers and outputs =, >, or <.

input Each test case has only one line containing two integers separated by a space. The first integer is in decimal representation and the second integer may be represented in decimal or English *There is always a blank line after every test case.*

output For each test case, output two lines: the first line contains =, >, or < depending on the comparison outcome *and a blank line as the second.*

Example:

Input

```
1 3

1 one

5 three

45 forty five

1001001100 one billion one million one thousand one hundred
```

Output

```
<

=

>

=

=
```

3 Is the number a cube?

A non-negative integer x is a cube if for some integer n, $x = n^3$. For example, 8 is a cube ($8 = 2^3$). 1 is also a cube ($1 = 1^3$). However, 2 is not a cube.

Write a program to decide if a given non negative integer is a cube or not.

input Each test case has one line containing a non negative integer x where $0 \leq x \leq 2^{31} - 1$. *There is always a blank line after every test case.*

output For each test case, output two lines: the first line containing either `CUBE` or `NOT CUBE`, where `CUBE` means the number is a cube according to our definition, and `NOT CUBE` means the number is not a cube; *and a blank line as the second*

Example:

Input

```
8

1

2

27

64

1331
```

Output

```
CUBE

CUBE

NOT CUBE

CUBE

CUBE

CUBE
```

4 A cube is a sequence sum

A positive cube $x = n^3$ can always be expressed as the sum of n odd numbers, where each number is the previous number plus 2 as shown in the following.

$$\begin{aligned} 1^3 &= 1 \\ 2^3 &= 3 + 5 \\ 3^3 &= 7 + 9 + 11 \\ 4^3 &= 13 + 15 + 17 + 19 \end{aligned}$$

Given an positive integer, decide if a given number is a cube or not. If it is a cube print the sequence sum expression (see above) and if it is not a cube print NOT CUBE.

input Each test case has one line containing a positive integer x where $1 \leq x \leq 2^{31} - 1$. *There is always a blank line after every test case.*

output For each test case, output two lines: the first line containing either sequence sum expression or `NOT CUBE`, where printing sequence sum means the number is a cube, and `NOT CUBE` means the number is not a cube; *and a blank line as the second*

Example:

Input

```
8

1

2

27

64

1331
```

Output

```
8 = 3 + 5

1 = 1

NOT CUBE

27 = 7 + 9 + 11

64 = 13 + 15 + 17 + 19

1331 = 111 + 113 + 115 + 117 + 119 + 121 + 123 + 125 + 127 + 129 + 131
```

5 Binary sequence

A binary sequence is a sequence of 0 and 1. For example, 00010111. By circular shift each bit to the right, from 00010111, we get 10001011, 11000101, 11100010, 01110001, 10111000, 01011100, 00101110. All these eight sequences are circular sequences of each other. One binary sequence x is a subsequence of another binary sequence y circularly if either x is one of the circular sequences of y or x is part of some circular sequence of y. For example, $x = 10001011$ is a subsequence of $y = 00010111$ circularly, as x is one of the circular sequences of y (see example above). As another example, $x = 101110$ is also a subsequence of the same sequence $y = 00010111$ circularly, as 101110 is part of 00101110, which is a circular sequence of y (see example above). Finally, $x = 1010$ is not a subsequence of $y = 00010111$ circularly.

Write a program that decides if the first binary sequence is a subsequence of the second binary sequence circularly.

input Each test case has only one line containing two binary sequences separated by a space. The maximum length of a sequence is 1024. *There is always a blank line after every test case.*

output For each test case, output two lines: the first line contains YES or NO depending on the subsequence checking *and a blank line as the second.*

Example:

Input

```
10001011 00010111

101110 00010111

1010 00010111
```

Output

```
YES

YES

NO
```

6 Binary De Bruijn sequence

A binary sequence y is a De Bruijn sequence of length k if every binary sequence of length k is a subsequence of y circularly exactly once. For example, $y = 00010111$ (the binary string used in the example of the previous problem) is a De Bruijn sequence of length 3. Here are all the binary sequences of length 3: 000, 001, 010, 011, 100, 101, 110, 111. Observe that 00010111, 00010111, 00010111, 00010111, 00010111 (after circular shift), 00010111, 00010111 (after circular shift), and 00010111.

Write a program that determines if a binary sequence is a De Bruijn sequence of length k when the binary sequence and k are provided.

input Each test case has only one line containing a binary sequence and an positive integer k separated by a space. *There is always a blank line after every test case.*

output For each test case, output two lines: the first line YES or NO depending on whether the binary sequence is a de bruijn sequence of length k or not *and a blank line as the second.*

Example:

Input

```
00010111 3

00010111 2

00010111 4

00011111 3

1111011001010000 4
```

Output

```
YES

NO

NO

NO

YES
```

7 Multiplication Row

Multiplication row is used to help students learn multiplication one number at a time. It has two rows and an example is show below.

```
x  1  2  3  4
3  3  6  9 12
```

The first row starts with the lower case x, which stands for multiplication and is followed by the integers 1 to n. In the example, n is 4. The second row starts with an integer, which is the number that we want to learn more about how it multiplies other numbers. This number is followed by the products of it multiplied by the corresponding integers in the first row.

Write a program to output a multiplication row. Note that all product column widths are the same and numbers in each column are line up with the least significant digit. Also, note the product column widths are as small as possible while leaving at least a space between digits. The first column has the smallest width where the number that we want to learn more about how it multiplies other numbers may fit. Note that the format of the output is important.

input Each test case has two lines. The first contains two positive integers. The first number is n, a positive integer, and the second number is the non-negative integer that we want to learn more about how it multiplies other numbers. The second is a blank line. *There is always a blank line after every test case.*

output For each test case, three lines: two lines for the Multiplication Row *and a blank line.*

Example:

Input

```
4 3

5 11
```

Output

```
x  1  2  3  4
3  3  6  9 12

x  1  2  3  4  5
11 11 22 33 44 55
```

8 Multiplication Table

A Multiplication Table is a generalization of the Multiplication Row (previous problem) so students may learn multiplication results for a group of values. Here is an example.

```
x  1  2  3  4
4  4  8 12 16
3  3  6  9
2  2  4
1  1
```

This Multiplication Table may not be the same that we used in grade school, but it captures the same information. This is the Multiplication Table for $n = 4$.

Write a program to produce a Multiplication Table of value n. The column widths and output format are similar to the Multiplication Row problem. Remember all digits in all columns are lined up with the least significant digit.

input Each test case has one positive integer in a line and this number refers to n in the problem description. *There is always a blank line after every test case.*

output For each test case, $n + 2$ lines: $n + 1$ lines for the Multiplication Table *and a blank line.*

Example:

Input

```
4

7
```

Output

```
x  1  2  3  4
4  4  8 12 16
3  3  6  9
2  2  4
1  1

x  1  2  3  4  5  6  7
7  7 14 21 28 35 42 49
6  6 12 18 24 30 36
5  5 10 15 20 25
4  4  8 12 16
3  3  6  9
2  2  4
1  1
```