University of Arkansas
Department of Computer Science and Computer Engineering
2014 High School Programming Contest
Problems

Wing Ning Li Hung Nguyen Adam Higgins Paul Martin
Sawyer Anderson Wes Deneke Tyler Moore

Revision Date: March 12, 2014

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 Divisible Checking

One integer is divisible by another integer if, after dividing, the remainder is 0. For example, 10 is divisible by 5 and 10 is also divisible by 2; 10 is not divisible by 3 and it is not divisible by 15. Since dividing by 0 is an error or illegal, in this problem, an integer divided by 0 is treated as not divisible.

Write a program to decide if one integer is divisible by another integer.

input Each test case has one line containing two integers x and y separated a space, where $-10,000 \leq x, y \leq 10,000$. *There is always a blank line after every test case.*

output For each test case, output two lines: the first line containing either `DIVISIBLE` or `NOT DIVISIBLE`, where `DIVISIBLE` means x is divisible by y, and `NOT DIVISIBLE` means otherwise; *and a blank line as second*

Example:

Input

```
10 5

10 2

10 3

10 15

10 0

```

Output

```
DIVISIBLE

DIVISIBLE

NOT DIVISIBLE

NOT DIVISIBLE

NOT DIVISIBLE
```

2 Primality Checking and Prime Factoring

A positive integer greater than 1 is said to be prime if it is only divisible by two integers: 1 and itself. For this problem, only integers satisfying that definition are considered a prime. For example, 5 is a prime (5 is divisible by only 1 and 5) and 6 is not a prime (6 is divisible by 1, 2, 3, and 6)

Any integer greater than 1 or less than -1 that is not a prime can be expressed as a product of primes or the negation of a product of primes. For example, $6 = 2 \times 3$, where 2 and 3 are primes.

Given an integer, print either `PRIME` or `NO PRIME FACTOR EXPRESSION` or the product of primes according to the format below.

input Each test case has only one line containing an integer x where $-1,000,000,000 \leq x \leq 1,000,000,000$. *There is always a blank line after every test case.*

output For each test case, output two lines: the first line is the product of primes that factor x or `PRIME` if x is a prime or `NO PRIME FACTOR EXPRESSION` if x cannot be expressed as a product of primes, *and a blank line as second.* You can assume that the prime factors y of x are in the range $-10,000 \leq y \leq 10,000$. The prime factors should be ordered in increasing values with appropriate powers (please see the sample output).

Example:

Input

```
3

12

192

139148064

0

-12
```

Output

```
PRIME

2^2 * 3

2^6 * 3

2^5 * 3^3 * 11^5

NO PRIME FACTOR EXPRESSION

-2^2 * 3
```

3 Sequence Checking

Let us consider sequences that consist of 0,1, and 2. For example, 00, 012, 20, 0101, 21010, 010 and 0102101 are such sequences. A sequence is called **repeat-free** if no subsequence repeats immediately after itself. For instance, 00 is not repeat-free because 0 is immediately followed by 0; 0101 is not repeat-free since 01 is immediately followed by 01; 21010 is not repeat-free because 10 is immediately followed by 10. The repeat-free sequences in the above examples list are 012, 20, 010, and 0102101.

Write a program to decide whether a given sequence of 0, 1, 2 is repeat-free or not.

input Each test case consists of a single line containing the sequence. *There is always a blank line after every test case.*

output For each test case, output either REPEAT-FREE or NOT REPEAT-FREE in a line *and a blank line.*

Example:

Input

```
00

012

```

Output

```
NOT REPEAT-FREE

REPEAT-FREE
```

4 Sequence Generation

Please refer to the previous problem for the descriptions of 0, 1, 2 sequences and repeat-free sequences. We may order sequences based on lexicographical order. In lexicographical order, we decide which of the two sequences is smaller by considering the first character of each sequence. If they are the same, we move to the next character of each sequence and continue until the considered characters are not equal or until the end of one sequence is reached. Then the sequence with a smaller considered character or reaching the end is smaller. The sequences in the previous problem statement can be sorted in decreasing lexicographical order:

```
00
010
0101
0102101
012
20
21010
```

Write a program to list all repeat-free sequences in decreasing order one by one until n sequences are listed.

input Each test case has one positive integer n in a line. n is the number of repeat-free sequences to list. *There is always a blank line after every test case.*

output For each test case, $n + 1$ lines: n lines for n repeat-free sequences and a blank line.

Example:

Input

```
1

4

8
```

Output

```
0

0
01
010
0102

0
01
010
0102
01020
010201
0102010
0102012
```

5 Caesar's Cipher I

A Caesar Cipher is a simple substitution cipher where each letter in the message (the plaintext) is encrypted using a uniformly shifted alphabet. With a shift of 3, 'A' becomes 'D', 'R' becomes 'U', and 'Z' becomes 'B'. All non-alphabetic characters (including punctuation, spaces, and numerals) are removed from the plaintext before constructing the ciphertext. All characters in the ciphertext should be lowercase.

Write a program which reads a plaintext in, along with an integer shift, and then encrypts that plaintext using that shift. Output the result of the encryption (the ciphertext).

input Each test case has 2 lines. The first line contains one non-negative integer n, a shift. The second line contains an English plaintext message as a string. *There is always a blank line after every test case.*

output For each test case, a single line containing the ciphertext of the message, *followed by a blank line.*

Example:

Input

```
3
Et tu, Brute?

27
ABCabc

```

Output

```
hwwxeuxwh

bcdbcd
```

6 Caesar's Cipher II

In order to increase the strength of the cryptosystem, Caesar has added an additional cryptographic operation to his simple substitution cipher (described in Caesar's Cipher I). First, all non-alphabetic characters are removed from the plaintext. Then each message is encrypted using the simple substitution cipher, and then encrypted further using a columnar transposition cipher. In order to perform the columnar transposition cipher the shifted plaintext (which is obtained using the cipher of the previous problem) is arranged in a table with a positive integer number of columns. To create the ciphertext, the table is read from top to bottom, column by column. Here's an example of a columnar transposition:

```
Number of shift 0
Number of columns: 5
Plaintext: This message will be encrypted.

thism
essag
ewill
beenc
rypte
d

Ciphertext: teebrdhsweyisiepsalntmglce
```

Write a program which reads a plaintext in, along with an integer shift and a column count, and then encrypts that plaintext using that shift and column count. Output the result of the encryption (the ciphertext).

input Each test case has 2 lines. The first line contains 2 space separated non-negative integers m, n, a shift m and a column count n. The second line contains an English plaintext message as a string. *There is always a blank line after every test case.*

output For each test case, a single line containing the ciphertext of the message, *followed by a blank line.* When a plaintext does not contain any alphabetic character or column count is 0, the ciphertext should be an empty line.

Example:

Input

```
3 4
Et tu, Brute?

27 1
ABCabc

```

Output

```
hehwuwxxw

bcdbcd
```

7 Point Clouds

A point cloud is a collection of data points, defined by X, Y, and Z coordinates, that are often used in computing applications to represent the external surfaces of 3D objects. A central problem in 3D modeling is calculating the distance between the members of point clouds.

Given a pair of 3D points, $X_1, Y_1, Z_1 and X_2, Y_2, Z_2$, the distance between these 2 points can be determined using the equation:

$$d = \sqrt{(X_1 - X_2)^2 + (Y_1 - Y_2)^2 + (Z_1 - Z_2)^2}$$

Write a program that accepts the integer coordinates ($-10,000 \leq x, y, z \leq 10,000$) of an arbitrary number 3D points (the total number of points is no more than 100) as input and outputs **the square of the max distance** between any pair of points.

input Each test case has several lines of input. The first n are the space delimited coordinates of all of the points in the point cloud, of the format: X Y Z. *There is always a blank line after every test case.*

output For each test case, output two lines: the first line is the square of the distance between the pair (or pairs) of points that are farthest apart *and a blank line as the second line.*

Example:

Input

```
0 0 0
1 1 1
2 2 2
3 3 3
4 4 4
5 5 5
6 6 6
7 7 7
8 8 8
9 9 9
```

Output

```
243
```

8 Cluster Extraction

In computer vision, having objects separated as individual clusters provides a mechanism for obtaining candidates that could potentially represent an object that is being searched for. A cluster is a collection of points that are all within some minimum **measure** of their nearest neighbor in a point cloud. Note that a single point is not considered a cluster.

Given a pair of 3D points, $X_1, Y_1, Z_1 and X_2, Y_2, Z_2$, the **measure** between these 2 points can be determined using the equation:

$$d = (X_1 - X_2)^2 + (Y_1 - Y_2)^2 + (Z_1 - Z_2)^2$$

Write a program that accepts the minimum **measure** and integer coordinates $(-10,000 \leq x, y, z \leq 10,000)$ of an arbitrary number 3D points (the total number of points is no more than 100) as input and outputs the number of clusters.

input Each test case has several lines of input. The first line specifies the value of the minimum **measure**. The next n lines specify the space delimited coordinates of all of the points in the point cloud, of the format: X Y Z. *There is always a blank line after every test case.*

output For each test case, output has 2 lines. When the number of clusters is greater than zero, the first line contains the number of clusters. If the number of clusters is zero, the first line prints `NO CLUSTERS`. *The final line output is a blank line.*

Example:

Input

```
3
0 0 0
1 1 1
2 2 2
3 3 3
4 4 4
5 5 5
6 6 6
7 7 7
8 8 8
9 9 9

```

Output

```
1
```