University of Arkansas
Department of Computer Science and Computer Engineering
2010 High School Programming Contest
Problems

Wing Ning Li Hung Nguyen Ebony Buckley Patrick Anderson
Mike Wilkins

Revision Date: March 12, 2010

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 2147483647.

A *string*'s maximum length is 1024.

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 Coprime: Checking

Two integers a and b are defined as coprime or relative prime if no positive divisor other than 1 exists that divides them. For example, 9 and 10 are coprime. Notice that neither 9 nor 10 is a prime number. A prime number is a natural number $(0, 1, 2, \ldots)$ of which the divisors are only 1 and itself. The numbers 6 and 9 are not coprime because they both are divisible by 3 and 1 is not the only divisor.

Write a computer program to determine if a pair of integers is coprime.

input each test case has a line containing two positive integers separated by one or more spaces. *There is always a blank line after every test case.*

output for each test case, output two lines: the first line containing `YES` if the two integers are coprime, and `NO` otherwise, *and a blank line as second.*

Example:

Input

```
9 10

6 9
```

Output

```
YES

NO
```

2 Coprime: Extension

The notion of two integers being coprime could be extended to a set of more than two integers in a natural way. A set of two or more integers is defined as coprime if every pair of integers in the set are coprime. The size of a coprime set is one less than the number of elements in the set.

For example, the set {9, 10} is coprime and the size is 1; the set {9, 10, 49} is coprime and the size is 2; whereas the set {9, 10, 35} is not coprime because not every pair are coprime, for instance 10 and 35 are not coprime.

Given a sequence of two or more integers, we need to determine the largest size of a coprime set that can be constructed using the integers in the sequence. For example, given 3, 6, 9, 14, the answer is 1 because {3, 14} is coprime; given 3, 6, 9, 12, the answer is 0 because no coprime set can be constructed; given 15, 3, 6, 5, 14, the answer is 2 because {3, 5, 14} is coprime.

input an input instance has two lines. The first line contains a positive integer, n, indicating the number of integers in the second line. The second line contains $n$integers separated by one or more spaces. *There is always a blank line after every test case.*

output for each test case, output two lines: the first line containing the largest *size* of a coprime set that can be constructed using the integers in the input line, *and a blank line as second.*

Example:

Input

```
4
3 6 9 14

4
3 6 9 12

5
15 3 6 5 14
```

Output

```
1

0

2
```

3 Periodic decimal sequence: Verifier

A decimal sequence is a finite sequence of decimal digits, which are 0, 1, 2, 3, 4, 5, 6, 7, 8, 9. We may use $a_1 a_2 \ldots a_n$ to represent a decimal sequence of size n, where each a_i, $1 \leq i \leq n$, is a decimal digit. For example, 119872345506 is such a sequence of length 12, where $a_1 = 1$, $a_2 = 1$, $a_3 = 9$, $a_4 = 8$, $a_5 = 7$, $a_6 = 2$, $a_7 = 3$, $a_8 = 4$, $a_9 = 5$, $a_{10} = 5$, $a_{11} = 0$, and $a_{12} = 6$. Two sequences are equal (=) if they have the same size and the digits in the corresponding positions are the same. A sequence is called periodic with respect to k if the following are true:

1. n is divisible by k, $k < n$
2. $a_1 \cdots a_n = a_1 \cdots a_k a_1 \cdots a_k \ldots a_1 \cdots a_k$. That is the whole sequence can be obtained by repeating the first k digits two or more times. For example, 789789789 is periodic with respect to 3. Notice that the whole sequence can be obtained by repeating 789 three times.

You are asked to write a program to verify for a given decimal sequence and positive integer k whether the sequence is periodic with respect to k or not.

input two lines per test case: the first line contains a decimal sequence, and the second contains a positive integer representing k. *There is always a blank line after every test case.*

output for each test case, output two lines: the first line containing `YES` if the input sequence is periodic with respect to k and `NO` otherwise, *and a blank line as second.*

Example:

Input

```
789789789
3

789788
2
```

Output

```
YES

NO
```

4 Periodic decimal sequence: period finder

The *period* of a periodic decimal sequence is the smallest positive integer k with respective to which the sequence is periodic. For example, the period of 789789789789 is 3, even though the sequence is periodic with respect to 6. For a sequence that is not periodic, its period is 0.

You are asked to write a program to determine the period of a decimal sequence.

input each test case has one line containing a decimal sequence. *There is always a blank line after every test case.*

output for each test case, output two lines: the first line containning an integer representing the period of the input sequence, *and a blank line as second.*

Example:

Input

```
789789789789

1234567
```

Output

```
3

0
```

5 Reporting structure hierarchy: determining the cost

In some organization, each person directly reports to one person except for the president of the organization who does not report to anyone. In addition, each person may have either exactly zero direct reports or exactly two direct reports. Notice that such a structure is possible if the number of people in the organization is odd.

The two persons that report to the same individual are distinguished by calling one the left report and the other the right report. Each direct report has an associated cost. The left reporting cost, which is a constant, is denoted by L, and the right reporting cost, which again is a constant, is denoted by R.

The reporting cost to the president of each person, x, is $f(x)$ and is defined recursively as follows:

1. $f(x) = 0$ if x is the president;
2. otherwise if x is the left report of y, $f(x) = L + f(y)$; and if x is the right report of y, $f(x) = R + f(y)$

The cost of a reporting structure is the sum of reporting costs to the president of those individuals to whom no one reports.

For example, we have a seven-person organization. Let us use 1 to 7 to represent the seven individuals. 1 has its left report 2 and right report 3, 3 has its left report 4 and right report 5, 4 has its left report 6 and right report 7, and 2, 5, 6, 7 have no one reporting to them. If $L = 1$ and $R = 2$, the cost of this reporting structure is $f(2) + f(5) + f(6) + f(7) = 1 + 4 + 4 + 5 = 14$. If $L = 2$ and $R = 1$, the cost of the same reporting structure would be $2 + 2 + 5 + 4 = 13$.

A different reporting structure of the seven-person organization would be: 1 has it left report 2 and right report 3, 3 has its left report 4 and left report 5, 5 has its left report 6 and right report 7, and 2, 4, 6, 7 have no one reporting to them. Again if $L = 1$ and $R = 2$, the cost of this reporting structure is $1 + 3 + 5 + 6 = 14$. If $L = 2$ and $R = 1$, the cost of this reporting structure is $2 + 3 + 4 + 3 = 12$.

Given a reporting structure and L and R, write a program to compute the the cost of the reporting structure. Integers from 1 to n are used as ID for individuals in an organization. Note that the first two examples have the same reporting structure, even though the input looks different.

input each test case starts with a line containing n, L, R. n is the number of individuals in the organization; L and R are the costs of the left and right reporting. There are n lines next, each containing three integers representing the individual, its left report, and its right report respectively. An individual who has no direct reports is identified by both the left and right report being 0. *There is always a blank line after every test case.*

output for each test case, output two lines: the first line with the cost as described above, *and a blank line as second.*

Example:

Input

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

7 1 2
7 0 0
4 6 7
1 2 3
2 0 0
5 0 0
6 0 0
3 4 5

5 1 8
1 0 0
5 2 3
3 0 0
2 0 0
4 1 5
```

Output

```
14

14

26
```

6 Reporting structure hierarchy: finding the cost of an optimal structure

Given n, L, and R as the size of an organization and the costs of the left and right reports respectively, an optimal reporting structure is a structure that has the least cost among all possible allowed structures (see the previous problem about the reporting structure).

Write a program to determine the cost of an optimal reporting structure among all possible reporting structures.

input each test case consists of one line containing three integers separated by one or more spaces representing n, L, R respectively. *There is always a blank line after every test case.*

output for each test case, output two lines: the first line containing an integer that is the cost of an optimal reporting structure, *and a blank line as second.*

Example:

Input

```
7 1 2

5 1 8

9 1 2
```

Output

```
12

19

17
```

7 Puzzles/Games: SUDOKU Checker

Sudoku is a popular fill-in-the-blanks puzzle using numbers from 1 to 9. A *Sudoku* puzzle consists of a 9x9 grid with each square that can be filled in by a number from set S = {1, 2, 3, 4, 5, 6, 7, 8, 9} provided that it satisfies all of the following rules:

1. Each row must contain each of the numbers 1 to 9 only once.
2. Each column must contain each of the numbers 1 to 9 only once.
3. Each block must contain each of the numbers 1 to 9 only once.

A block is a 3x3 grid, and a *Sudoku* puzzle consists of nine blocks, of which no two blocks may overlap each other. The rules ensure that the solution of the puzzle must have all elements of set S in each row, each column, and each block. Below is an example of a puzzle and its solution:

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

→

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

A solution not satisfying all of the above rules is considered incorrect. Given a proposed puzzle solution, write a program to check if the proposed solution is correct or not.

input each test case consists of nine lines; each line contains nine digits representing the numbers in the corresponding row. *There is always a blank line after every test case.*

output for each test case, output two lines: the first line containing either `CORRECT` or `INCORRECT` for the correct and incorrect proposed solution respectively, *and a blank line as second.*

Example:

Input

```
827536941
516849327
943172658
271394865
365718294
498625173
652483719
189257436
734961582

123456789
234567891
345678912
456789123
567891234
678912345
789123456
891234567
912345678

532146785
146789253
789253146
213465897
457891362
698327415
325614978
861972534
974538621
```

Output

```
CORRECT

INCORRECT

INCORRECT
```

8 Puzzles/Games: SUDOKU solver

Write a program to solve a *Sudoku* puzzle. Refer to the "Sudoku Checker" problem above for the rules of the game. You may assume that every puzzle to be solved by your program is guaranteed to be solvable with a unique solution.

input each test case consists of nine lines; each line contains nine characters from 1, 2, 3, 4, 5, 6, 7, 8, 9, and *. A blank square is indicated by the character *. *There is always a blank line after every test case.*

output for each test case, output ten lines: the first nine lines contain the solution to the given puzzle with one line representing one row, *and a blank line as tenth.*

Example:

Input

```
8**5*6*4*
51*******
943**265*
**1**4865
***7*8***
4986**1**
*524**719
*******36
*3*9*1**2
```

Output

```
827536941
516849327
943172658
271394865
365718294
498625173
652483719
189257436
734961582
```