University of Arkansas
Department of Computer Science and Computer Engineering
2011 High School Programming Contest
Problems

Wing Ning Li   Hung Nguyen   Adam Higgins

Revision Date: March 9, 2011

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 Moving a Rock by One Square

Alice and Bob are playing a game involving moving a rock through some adjacent squares. The rules of the game are as follows:

1. There are N 1×1 squares adjacent to each other to form a $1 \times N$ rectangle strip. The squares are numbered from 1 to N from left to right.
2. The rock is initially at square 1.
3. When a player's turn comes, s/he has to move the rock to the next square, e.g. from square i to square $i + 1$ for $1 \leq i < N$. When the rock is currently at square N, in the next turn it will be moved to square 1, this is called wraparound.
4. The number of turns is T and the game will be finished after T turns.
5. Alice and Bob alternate their turns in playing the game. Alice always starts first.
6. After T turns, if the rock is in an odd numbered square, Alice wins. Otherwise, Bob wins.

Given N squares and T turns, write a program to determine who will win.

input Each test case has a line containing two positive integers separated by one or more spaces. The first integer is N, the number of squares. The second integer is T, the total number of turns. The range for N and T is $2 \leq N, T \leq 10,000$. *There is always a blank line after every test case.*

output For each test case, output two lines: the first line containing `Alice wins` if Alice will win the game, or `Bob wins` otherwise, *and a blank line as second.*

Example:

Input

```
5 4

5 6

```

Output

```
Alice wins

Bob wins
```

2 Moving a Rock by X or Y Squares

In the previous problem, the game is played by moving the rock to the next square with wraparound. The game in this problem is similar to the game in the previous problem except that an integer $X \geq 0$ is associated with even numbered squares and an integer $Y \geq 0$ is associated with odd numbered squares. If the rock is currently in an even square, the next player should move the rock X square(s) to the right with wraparound; and if the rock is currently in an odd square, the next player should move the rock Y square(s) to the right with wraparound. Notice that the game in the previous problem is a special case of this game where $X = 1$ and $Y = 1$.

The input of the problem consists of N (the number of squares), T (the number of turns), X (the number of moves from an even square), and Y (the number of moves from an odd square). Given an input to the problem, write a program to determine who will win.

input Each test case has a line containing two positive integers followed by two nonnegative integers. Integers are separated by one or more spaces. The four integers are N T X Y as discussed in the problem description. The range for N and T is $2 \leq N, T \leq 10,000$, and the range for X and Y is $0 \leq X, Y \leq 50$. *There is always a blank line after every test case.*

output For each test case, output two lines: the first line containing `Alice wins` if Alice will win the game, or `Bob wins` otherwise, *and a blank line as second.*

Example:

Input

```
5 4 1 1

5 4 2 1

```

Output

```
Alice wins

Bob wins
```

3 Message Signature Creator

When we send a message over the internet, sometimes the message does not get delivered exactly as we hope. To help the receiver verify the accuracy of a message, we include the message signature at the end of the message. For this problem, a message contains only English alphabet's letters: a-z and A-Z, commas, periods, and spaces. Each character has a numeric value associated with it, which is the ASCII number for that character:

1. a-z have numeric values in the range 97-122. For example, a is 97, b is 98, c is 99, ..., z is 122.
2. A-Z have numeric values in the range 65-90. For example, A is 65, B is 66, C is 67, ..., Z is 90.
3. Comma (,) has numeric value 44.
4. Period (.) has numeric value 46.
5. Space ( ) has numeric value 32.

A message's signature is the sum of all the numeric values of the characters contained in a message. Given a message, write a program to compute its signature.

input Each test case has only one line containing the message using the specified characters. The message will not have more than 5,000 characters. *There is always a blank line after every test case.*

output For each test case, output two lines: the first line containing the signature as an integer, *and a blank line as second.*

Example:

Input

```
ABaab

Hello, world.

Hi, Alice. This is Bob.

```

Output

```
423

1174

1822
```

4 Message Repairer

This problem is closely related to the previous one. Given the message and its signature, the receiver not only can verify the accuracy of most messages but also can, in some cases, reconstruct the original ones.

Suppose that during transmission, the only possible error is to have two characters swapped. That is, all instances of character a_1 in the message become a_2, which is also in the same message, and vice versa. For example, aabac becomes bbabc, where a and b get swapped due to error. Our task is to reconstruct the original message from the modified message and the signature of the original message. You can assume that each test case has a unique solution, and the message always has an error.

input Each test case has two lines. The first line is the message that has the error described. The second line is the signature of the original (correct) message. The message will have at least 3 and no more than 5,000 characters. *There is always a blank line after every test case.*

output For each test case, output two lines: the first line containing the original message, *and a blank line as second.*

Example:

Input

```
AaBBb
423

Heool, wlrod.
1174

Hi, Ali.ec This is Bobc
1822

```

Output

```
ABaab

Hello, world.

Hi, Alice. This is Bob.
```

5 Grid 2-Coloring: Adding a Color to the Partial 2-Coloring

In a grid 2-coloring, each grid cell is assigned either a *RED* or a *BLUE* color. If not all the cells are assigned a color yet, the current 2-coloring is called a partial 2-coloring. A grid 2-coloring or partial 2-coloring is valid if no monochromatic square exists for the current grid coloring, that is no square exists such that the four corner cells of the square have the same color. Figure 1 shows a valid partial 2-coloring of a 4×4 grid, and Figure 2 shows an invalid partial 2-coloring of a 4×4 grid. Given a valid partial 2-coloring of an $N \times N$ grid and a new color for a given cell that is not colored, write a program to decide if coloring the cell with the given color results in a valid 2-coloring.

Figure 1:

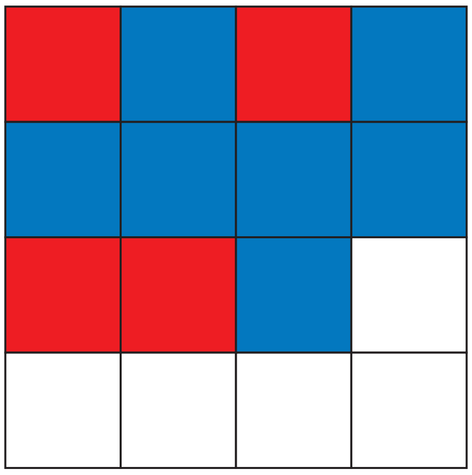
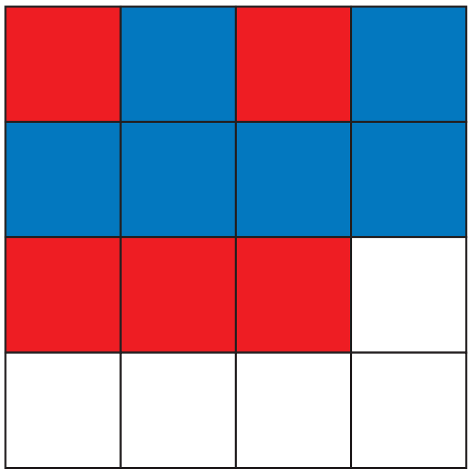


Figure 2:



input Each test case has several lines. The first line is an integer N, $2 \leq N \leq 9$, indicating the size of the grid; the next N lines representing a valid partial 2-coloring, each line has N characters with `R` standing for *RED* color, `B` for *BLUE* color, and `*` for not-yet-colored cell. The last line contains two integers and a character separated by spaces. The two integers are the row and column of the cell to be colored, where rows and columns are numbered from left to right and top to bottom, respectively. The first row or column is numbered 1. The character is the color of the cell added. *There is always a blank line after every test case.*

output For each test case, output two lines: the first line containing either `YES` or `NO`, where `YES` means a valid (partial) 2-coloring and `NO` means an invalid (partial) 2-coloring; *and a blank line as second.*

Example:

Input

```
4
RBRB
BBBB
RB**
****
3 3 R

4
RBRB
BBBB
RB**
****
3 3 B

3
RRR
BB*
***
3 3 B
```

Output

```
NO

NO

YES
```

6 Grid 2-Coloring: Can This Valid Partial 2-Coloring Extend to a Full Valid 2-Coloring?

Given a valid partial 2-coloring of an $N \times N$ grid, decide if the valid partial 2-coloring may be extended to a full and valid 2-coloring. That is, without making changes to the already-colored cells, can we color all of the not-yet-colored cells in *RED* or *BLUE* color so the resulting grid 2-coloring is valid?

input Each test case has several lines. The first line is an integer N, $2 \leq N \leq 9$, indicating the size of the grid. The next N lines representing a valid partial 2-coloring. Each line has N characters with `R` standing for *RED* color, `B` for *BLUE* color, and `*` for not-yet-colored cell. *There is always a blank line after every test case.*

output For each test case, output two lines: the first line containing either `YES` or `NO`, where `YES` means the valid partial coloring can be extended to a valid 2-coloring, and `NO` means the valid partial 2-coloring can not be extended to a valid 2-coloring; *and a blank line as second.*

Example:

Input

```
4
RBRB
BBBB
RB**
****

3
RRR
BB*
***

```

Output

```
NO

YES
```

7 Does This Point Lie on This Line?

The slope-intercept equation for a line in a plane is $y = Ax + B$. Given a point (x, y) and a line equation specified by A, B, write a program to determine whether (x, y) lies on that line or not. Assume that A, B, x, y are all integers, and there is no vertical line.

input Each test case has two lines. The first line contains A and B parameters of a line equation separated by spaces. The second line contains x and y coordinates of some point separated by spaces. The input range is $-20,000 \leq A, B, x, y \leq 20,000$. *There is always a blank line after every test case.*

output For each test case, output two lines: the first one with either `YES` or `NO` to indicate whether point (x, y) lies on line $y = Ax + B$ or not, *and a blank line as second.*

Example:

Input

```
1 1
0 0

1 1
1 2

```

Output

```
NO

YES
```

8 Finding the Line Equation

Given two different points (x_1, y_1) and (x_2, y_2) not vertically aligned, your task is to find the slope-intercept line equation $y = Ax + B$ for the line that goes through the two points. Assume that the coordinates of the input points are integers. In case the value of A or B is a rational number, the reduced form is required. A rational number $\frac{a}{b}$, with both a and b as integers, is in its reduced form if a and b do not have any common integer divisors other than 1. For example, the reduced form of $\frac{4}{8}$ is $\frac{1}{2}$.

input Each test case has two lines. The first one contains two integers separated by spaces for x_1 and y_1. The second contains x_2 and y_2 in a similar fashion. The range for the input is $-50,000 \leq x_1, y_1, x_2, y_2 \leq 50,000$. *There is always a blank line after every test case.*

output For each test case, output two lines. The first line contains A and B separated by one space, which are the parameters of the line equation of the line that passes through the two input points. If A or B is a rational number with reduced form $\frac{a}{b}$ where $b > 1$, print it as `a/b`. If $a = 0$ or $b = 1$, only output a. For this problem, a and b will never be greater than $2^{31} - 1$ or less than -2^{31}. *The second line is a blank line.*

Example:

Input

```
1 2
0 1

5 6
1 8

```

Output

```
1 1

-1/2 17/2
```