Irish Collegiate Programming Competition 2015

Problem Set

University College Cork ACM Student Chapter

March 28, 2015

Instructions

Rules

- All mobile phones, laptops and other electronic devices must be powered off and stowed away for the duration of the contest.
- The only networked resource that teams are permitted to access is the submission system.
- If a team discovers an ambiguity or error in a problem statement they should tell an organiser. If the organisers agree that an ambiguity or error exists, a clarification will be issued to all teams.
- No multi-threading is allowed, and no sub-processes.
- No file input/output is allowed. All input to your program will be provided via standard input (stdin) and all output should be printed to standard output (stdout). Examples of how to do this in each of the languages is provided in the resources section of the submission site.

Submission Instructions

- The submission system URL is: `http://4c245.ucc.ie:8080/`
- Your password will be provided by the organisers. Please notify an organiser if you are unable to log in.
- Submissions should consist of a single source file, **not a compiled executable**.
- To submit, click the "Submit a Solution" link, complete the submission form, upload your source file, and click save.
- Your submission should now be listed in your submission queue.
- Java solutions should be a single source file and should not include the package header. The testing script will also be renaming your file to `Pn.java` (where n is the problem number) and as such the main class for problem 1 should be defined as: `public class P1 {...}` and will be called with: `java P1 < input-file`

- C and C++ submissions will be compiled with the flags `-lm -O2`. `C#` submissions will be run using mono since the test environment is running on linux.

Testing and Scoring

- Solutions should be submitted in the form of source code, which will be compiled on the test environment. A solution will be accepted if it compiles on the test environment and produces the correct output for the sample inputs given in the question.
- The output from your program should be terminated by a new line and should not contain any additional whitespace at the beginning or end of lines.
- A solution will be tested against 10 separate test cases which are designed to test the limits and corner cases of the inputs. One point is given for each correct output.
- Programs are limited to one second of CPU time and 256MB of RAM.
- If a solution is submitted while an earlier solution to the same problem is still in the queue, the earlier submission will not be tested. The earlier submission will be marked with the message: "Old, not going to be tested".
- In the event of a tie, the winning team will be the one who's last scoring submission was submitted earliest.

1 Margaret's Minute Minute Manipulation

Margaret has always been a good maths student. She has been trying to apply the principles of binary quantum refraction to time travel in her free time. By encoding time in a binary format and adding a non negative time difference, δ, she is hoping to create a singularity in the fabric of space and time allowing one to jump by the amount δ into the future.

Time's usual representation is the well known *24h format* - e.g. 03:14:15. Although there is several possible ways to represent time in a binary form, the convention used throughout this exercise is as follows.

	H	**H**	**M**	**M**	**S**	**S**	
Binary Clock Format	0	0	0	0	0	0	8
	0	0	0	1	0	1	4
	0	1	0	0	0	0	2
	0	1	1	0	1	1	1
	0	3	1	4	1	5	

Figure 1: Binary Clock Format of time 03:14:15.

A 4×6 matrix can be used to represent time in a binary format. Each decimal digit of the *24h format* is encoded separately using 4 bits. The decimal digits are encoded in binary with the most significant bit on top, and the least significant at the bottom. For instance, the decimal number 3_{10} can be represented as 0011_2 in a 4-digit binary format, i.e. $(0 \times 2^3) + (0 \times 2^2) + (1 \times 2^1) + (1 \times 2^0) = 3$.

Provided a time of day $\mathcal{T}$ and a time difference δ, both in the Binary Clock format, you are to compute the time of day resulting from their summation, i.e. $\mathcal{T} + \delta$.

Input The first 4 lines represent the time of day and the subsequent 4 lines represent the time delta. Both clocks are guaranteed to be a valid time ranging from 00:00:00 to 23:59:59 inclusively. We further assume a *naive* implementation of time in which we do not care about time zones, leap seconds, nor the shifting effects of daylight saving time.

Output The output should consist of the resulting time $(\mathcal{T} + \delta)$ in the 4×6 matrix Binary Clock format. This should immediately be followed by a newline.

Sample Input 1

```
0 0 0 0 0 0
0 1 0 0 0 0
0 0 0 0 0 0
1 0 0 0 0 0
0 0 0 0 0 0
0 1 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 0
```

Sample Output 1

```
0 1 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 0
1 0 0 0 0 0
```

Sample Input 2

```
0 0 0 0 0 0
0 0 0 1 0 1
0 1 0 0 0 0
0 1 1 0 1 1
0 0 0 0 0 0
0 1 1 1 1 1
0 1 0 0 0 0
0 0 1 1 1 1
```

Sample Output 2

```
0 0 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 0
1 0 1 0 1 0
```

2 Ethel's Encryption

Ethel suspected that the North Side Alliance (NSA) were spying on her communications with her sister Lucy. In order to thwart the NSA's attempts, Ethel encrypted all of their communications using a Caesar cipher.

A Caesar cipher is an encryption method where each letter in the plaintext is shifted by an offset in order to produce the encrypted text. For example, given the offset value of 2, A becomes C, B becomes D, Y becomes A, and so on.

Your task is to decrypt Ethel's encrypted messages given that the offset is calculated by a^b.

Input The first line contains three integers n, a, and b. n is the number of characters in the encrypted message, including spaces. The numbers a and b are used to calculate the offset a^b, with $0 \leq a \leq 2^{31}$ and $0 \leq b \leq 2^{16}$, however at least a or b will be greater than 0 in each input.

The second line contains n characters representing the encrypted message. It is guaranteed to only contain upper-case letters (A-Z) or spaces, and is immediately followed by a newline.

Output The output should consist of a single line with the decrypted message in upper-case characters. Note that space are preserved between the encrypted and decrypted messages. This should immediately be followed by a newline.

Sample Input 1

```
11 3 3
IFMMP XPSME
```

Sample Output 1

```
HELLO WORLD
```

Sample Input 2

```
39 2 5
GT OTBKYZOMGZOUT UL ZNK RGCY UL
ZNUAMNZ
```

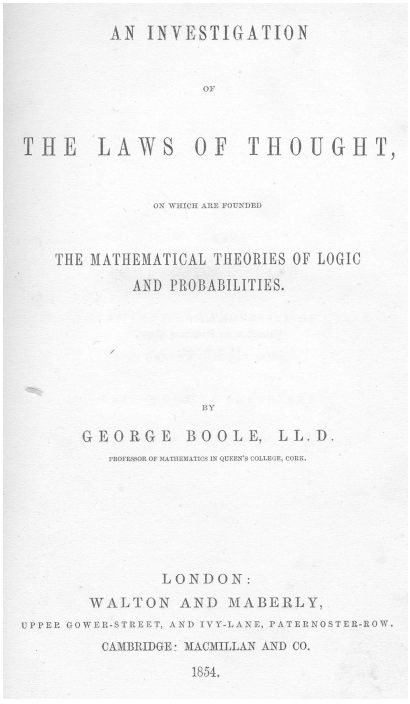
Sample Output 2

```
AN INVESTIGATION OF THE LAWS OF
THOUGHT
```

3 Boolean Postfix

Logical Boolean expressions are typically represented using *infix* notation, where the operators $(\wedge, \vee)$ are placed between the operands. For example, $((a \wedge b) \vee \neg c)$ states that for the expression to be *true*, both a and b should be *true*, or c should be *false*. In her mathematics, Mary Lucy Margret uses an alternate form, where the operator is placed after the operands, called *postfix* notation. For example, the above expression could be written in postfix notation as: $(a\ b \wedge c \neg \vee)$.

Your task is to write a program to parse and evaluate Mary Lucy Margret's Boolean expressions, which are represented in postfix notation. You can be confident in her mathematics; you are guaranteed to be given correct and valid expressions.


AN INVESTIGATION

OF

THE LAWS OF THOUGHT,

ON WHICH ARE FOUNDED

THE MATHEMATICAL THEORIES OF LOGIC AND PROBABILITIES.

BY

GEORGE BOOLE, LL.D.

PROFESSOR OF MATHEMATICS IN QUEEN'S COLLEGE, CORK.

LONDON:
WALTON AND MABERLY,
UPPER GOWER-STREET, AND IVY-LANE, PATERNOSTER-ROW.
CAMBRIDGE: MACMILLAN AND CO.
1854.

Figure 2: *An Investigation of the Laws of Thought* by George Boole, Professor of Mathematics in Queen's College, Cork, published 1854.

Input The first line of input contains a single integer T representing the number of expressions. Each of the next T lines contains a single Boolean formula in postfix notation. The line starts with a single integer n representing the number of tokens, with the remainder of the line containing n tokens, each separated by a single space.

The tokens 1 and 0 are used to denote Boolean truth values *true* and *false* respectively, and uppercase characters are used to denote the operators. Thus, the set of possible tokens are:

`1` for Boolean *true*,	`A` for logical *and*,	`X` for logical *exclusive-or*,
`0` for Boolean *false*,	`R` for logical *or*,	`N` for logical negation.

Output The output should consist of T lines where each line contains a 1 if the corresponding expression evaluates to *true*, or 0 otherwise.

Sample Input 1

```
3
3 0 1 R
3 0 0 R
7 1 0 A 1 R N N
```

Sample Output 1

```
1
0
1
```

Sample Input 2

```
4
9 0 0 A 0 N 0 N A R
9 0 1 A 0 N 1 N A R
9 1 0 A 1 N 0 N A R
9 1 1 A 1 N 1 N A R
```

Sample Output 2

```
1
0
0
1
```

4 Wonowon

There is a little village in northern Canada called Wonowon, its name coming from the fact that it is located at Mile 101 of the Alaska Highway. While passing through this village, a wandering mathematician had an idea for a new type of number, which he called a wonowon number. He defined a wonowon number as a number whose decimal digits start and end with 1, and alternate between 0 and 1. Thus, the first four wonowon numbers are 101, 10101, 1010101, 101010101.

Neither 2 nor 5 can divide any wonowon number, but it is conjectured that every other prime number divides some wonowon number. For example, 3 divides 10101 (i.e. 3×3367), 7 divides 10101 (i.e. 7×1443), 11 divides 10101010101010101010 (i.e. $11 \times 918273645546372819 1$).

Assume throughout that this conjecture is true, and let $W(p)$ denote the number of digits in the smallest wonowon number divisible by p. Thus, for example, $W(3) = 5$, $W(7) = 5$, $W(11) = 21$, $W(13) = 5$, $W(17) = 15$, $W(19) = 17$.

It has been found experimentally that for many primes p, $W(p) = p - 2$ (as in the case for $p = 7, 17, 19$). Thus, your task is to write a program which reads an integer n and outputs the number of primes for which $W(p) = p - 2$. Note that p cannot be 2 nor 5, and p is a prime number less-than or equal to n.

Input The input consists of a single integer $3 \leq n \leq 10000$.

Output The output should consist of a single integer representing the number of primes p for which $W(p) = p - 2$.

Sample Input 1

```
20
```

Sample Output 1

```
3
```

Sample Input 2

```
100
```

Sample Output 2

```
14
```

5 Alicia's Afternoon Amble

Alicia is staying in a hotel on the edge of town. She sets out in the morning with a list of landmarks to visit all across the city. This day of sightseeing will take her through the city, visiting a number of locations, all the way to the far-side of city where she will pause for lunch at the prestigious Pete's Polygon Pizza Parlour. Anything she misses must be visited on the return trip to the hotel in the evening.

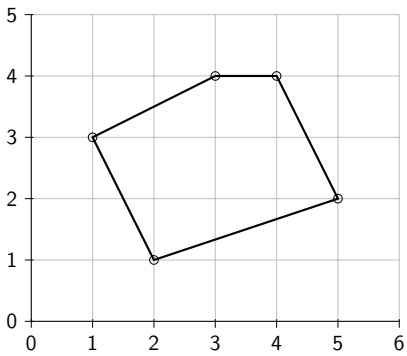


Figure 3: The locations and optimal solution for sample input 1.

Given the set of locations that Alicia would like to visit, find the length of the shortest tour which starts at her hotel, visits each of the locations, and returns back to the hotel. Beside the starting location, each location should be visited exactly once.

The starting hotel will be located at the leftmost x-coordinate. From there, the optimal path should visit locations at strictly increasing x-coordinates until Pete's Parlour is reached at the rightmost x-coordinate. From here, the optimal return path should visit any and all unseen locations in strictly decreasing x-coordinates. Note that the x-coordinate of each location will be unique.

Input The (x, y) coordinates of all locations are given as integers on a Euclidean plane. The first line of input contains a single integer n, $1 \leq n \leq 1000$, representing the number of locations. Each of the next n lines contains two integers x and y, $0 \leq x, y \leq 100000$, representing the coordinates of the n locations.

Output The output should consist of a single decimal containing the the total length of the tour rounded to two decimal places. This should immediately be followed by a newline.

Sample Input 1

```
5
1 3
2 1
3 4
4 4
5 2
```

Sample Input 2

```
10
4 1
13 4
21 3
25 9
28 10
42 1
43 2
50 4
67 10
68 9
```

Sample Output 1

```
10.87
```

Sample Output 2

```
131.65
```