Irish Collegiate Programming Competition 2014

Problem Set

University College Cork ACM Student Chapter

March 29, 2014

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 Binary Addition

In this problem, we study the simple task of summing multiple integers which are represented in their binary form. Integers are represented in binary by a series of 0s and 1s. Each digit represents an increasing power of 2 where the rightmost digit will be 2^0, followed by 2^1, 2^2, and so on. As an illustration, below are the first eight binary numbers with four binary bits and their corresponding decimal values:

Binary	Decimal
0000	0
0001	1
0010	2
0011	3

Binary	Decimal
0100	4
0101	5
0110	6
0111	7

A binary number can be converted into decimal by multiplying the binary bit by its respective power of 2. For example, the binary number 1101 can be converted into decimal 13 as follows:

$$1101_2 = 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0$$
$$1101_2 = 1 \times 8 + 1 \times 4 + 0 \times 2 + 1 \times 1$$
$$1101_2 = 13_{10}$$

Your task is to read a list of integers represented in binary, sum their values, and print the resulting binary number.

Input The first line contains an integer N; $2 \leq N \leq 1,000,000$; represented in binary. Each of the next N lines contain a single binary number which are to be added together. All binary numbers of the inputs are guaranteed to be exactly 32 binary characters in length, therefore can be stored in a 32-bit integer, and will be positive.

Output The output of your program should be a single line with the value representing the total sum of all N binary numbers. This number should be printed in binary notation with exactly 32 characters and should be immediately followed by a single newline character.

Sample Input 1

```
00000000000000000000000000000010
00000000000000000000000000000010
00000000000000000000000000000101
```

Sample Input 2

```
00000000000000000000000000000101
00000000000000000000000000001101
00000000000000000000000000000001
00000000000000000000000000000110
00000000000000000000000000000101
00000000000000000000000000001111
```

Sample Output 1

```
00000000000000000000000000000111
```

Sample Output 2

```
00000000000000000000000000101000
```

2 Look and Say

A 'Look and Say' sequence consists of a series of lines where each line *describes* the previous one; beginning with a starting sequence. Lines are described by counting continuous occurrences of characters. For example:

Line	Description	Sequence
1	starting sequence	1
2	one-1	11
3	two-1s	21
4	one-2, one-1	1211
5	one-1, one-2, two-1s	111221
6	three-1s, two-2s, one-1	312211
	...	

Your task is to implement a program to compute a 'Look and Say' sequence. You will be given the first line as input and asked to output the n^{th} line of the sequence.

Input The input is a single line consisting of the starting string sequence, s; followed by a space; and an integer $1 \leq n \leq 100$ which is the index of the line that should be calculated. Sequences for all inputs and expected outputs are guaranteed to only contain digits $0..9$ and have a length between $1 \leq |s| \leq 1000$. Therefore, the sequence should not be stored as an integer.

Output The output should consist of a single line containing the string representation of the n^{th} line of the sequence. Characters of the sequence should not be separated by any space and should be terminated by a new line character.

Sample Input 1

```
1 7
```

Sample Output 1

```
13112221
```

Sample Input 2

```
1234 4
```

Sample Output 2

```
13211231133114
```

3 Baffling Boolean Bulbs

In their latest contracts, Spiteful Architects Inc. decided to save money on light switches by only having switches in a single central control. Unfortunately, they also chose to save money by not installing windows or cameras. Therefore, the lighting manager has no way of seeing which lights are on or off.

However, he has been given the schematics for the lighting circuits. Between the light switches and the light bulbs is a complicated network of boolean logic gates.

This network consists of both switches as well as light bulbs, and some number of Boolean logic gates which are connected in between. The signal from switches and gates are fed through the network until they reach a bulb.

The set of Boolean gates consists of AND, OR, XOR, NOR, and NAND. Each of these gates takes two input signals and outputs a single signal. These signals are either an *ON* signal, represented by 1, or an *OFF* signal, represented by 0. Gates will take inputs from switches or other gates, and the output will connect to a gate or a light bulb. Output from a gate or switch can be used as input multiple times.

Finally the bulbs take a single input. If the bulb receives an *ON* signal, the bulb will light up. If it receives an *OFF* the bulb will remain unlit. Bulbs do not output signals themselves.

With the knowledge of which switches are turned-on and the particular lighting schematics, the unfortunate lighting manager sets out to determine which lights are on and which lights are off.

Input The first line contains three integers $2 \leq L \leq 5000$; $0 \leq B \leq 50000$ and $0 \leq G \leq 50000$. L represents the number of light switches, B is the number of light bulbs, and G represents the number of gates in the network.

Each of the next L lines contain two integers i and s. The first integer i, $1 \leq i \leq (L+B+G)$, is the identifier for a light switch. All identifiers are unique across the switches, bulbs, and gates. The second integer $s \in \{0,1\}$, is the state of the gate, with 1 representing *ON* and 0 representing *OFF*.

Each of the next B lines contain two integers i and j. The first integer i, $1 \leq i \leq (L+B+G)$, is the identifier for a light bulb. The second integer j, $1 \leq j \leq (L+B+G)$, is the id of the input signal source for the light bulb.

The next G lines contain four integers representing a gate in the circuit: i, j, k, and t, with $1 \leq i,j,k \leq (L+B+G)$ and $t \in \{1,2,3,4,5\}$. i is the identifier of the gate, j and k are the identifiers of the input signal source of the gate. Finally, t represents the type of the gate. Table 1 lists the types of gates and their respective Boolean truth-tables.

Gate Type		1	2	3	4	5
Input		AND	OR	XOR	NOR	NAND
0	0	0	0	0	1	1
0	1	0	1	1	0	1
1	0	0	1	1	0	1
1	1	1	1	0	0	0

Table 1: Output values for logic gates

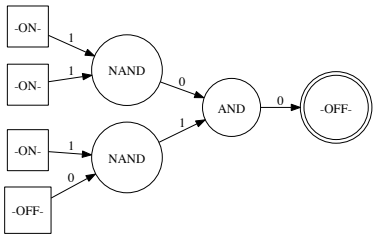


Figure 1: Sample Input 1

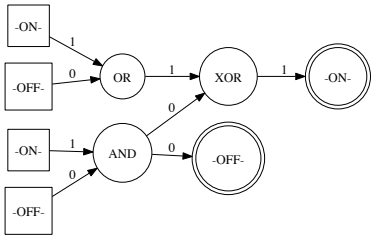


Figure 2: Sample Input 2

Output The output consists of a single line of space separated integers, representing the state of the lights in the circuit. 0 represents a light in the *OFF* state, and 1 a light in the *ON* state. The lights should be sorted in ascending order according to their identifier. As before, the output should immediately be followed by a single newline character.

Sample Input 1

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

Sample Output 1

```
0
```

Sample Input 2

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

Sample Output 2

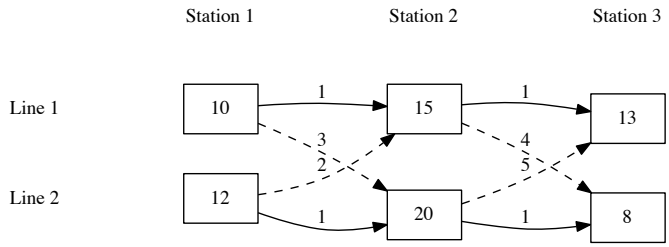
```
1 0
```

4 Assembly Line

Consider two parallel assembly lines producing identical products. Each assembly line is composed of a number of stations that contribute to the overall assembly of the end-product. Both assembly lines contain the same stations in the same order, however differences in abilities and experience of workers in each station, mean that the time to pass through each station is different.

Since the assembly lines run parallel to each other, it is possible to either move the output of station i on one assembly line to station $i+1$ on the other assembly line or to just continue to station $i+1$ on the same line.

Given the times to pass through each station and to move between stations, your task is to choose the sequence of stations which should be used to complete the end-product the fastest.



The instance above corresponds to sample input 1. The most efficient route consists of starting at station 1 on line 1, continuing to station 2 on line 1, then moving to station 3 on line 2 for a total time of $10+1+15+4+8=38$.

Input

The first line of input contains a single integer $1 \leq n \leq 10000$, the number of stations on both assembly lines. Each of the next n lines contain two positive integers t_{i1} and t_{i2}, where t_{i1} is the time that station i on assembly line 1 takes to complete its task, and t_{i2} is the time that station i on assembly line 2 takes to complete the same task.

The final $n-1$ lines each contain four positive integers m_{i1}, m_{i2}, c_{i1}, and c_{i2} where:

m_{i1} is the time to move from station i on line 1 to station $i+1$ on the same line,

m_{i2} is the time to move from station i on line 2 to station $i+1$ on the same line,

c_{i1} is the time to move from station i on line 1 to station $i+1$ on line 2, and

c_{i2} is the time to move from station i on line 2 to station $i+1$ on line 1.

Output

The output should consist of a single line containing n integers representing the stations which are to be manned. The i^{th} integer in the output should be 1 (2) if the i^{th} station of assembly line 1 (2, respectively) is to be manned.

Sample Input 1

```
3
10 12
15 20
13 8
1 1 3 2
1 1 4 5
```

Sample Output 1

```
1 1 2
```

Sample Input 2

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

Sample Output 2

```
1 2 1 2 2 1
```

5 Perfect Pancakes Preparation

Some say cooking is an art, but for John Dough cooking is merely the science of getting proportions right. Pancake recipes consistently involve the same three ingredients: eggs, milk, and flour. Of course, hundreds of accurate recipes can be found in cook books or even by asking grandmothers, but no guarantee is given if one does not respect the proportions specified in these recipes.

Figure 3 plots the proportion of flour and milk units per egg, for various recipes gathered from cook books. From the lower left to the upper right, the number of eggs in the recipes decreases (as there is more and more flour and milk). On the upper left corner we have lots of flour, and on the lower right corner, more milk. One can think of the space within the polyhedron as a *safe* region for exploring new pancake recipes.

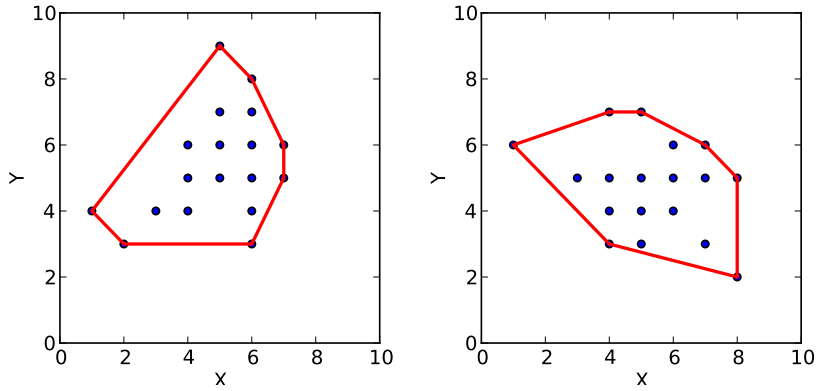


Figure 3: The x-axis plots the units of flour per egg; while the y-axis captures the units of milk per egg for multiple pancake recipes gathered by John Dough. The polyhedron is described by the pancake recipes for which flour and milk to egg proportions are considered extreme. The figure on the left corresponds to sample instance 1 and similarily the figure on the right to sample instance 2.

John Dough conjectures that any combination of milk and flour units per egg lying within the polyhedron is safe to explore since it does not go outside the safe proportions.

You are given the (x, y) coordinates of each pancake recipe. Your task is to write a program to identify the boundary points (recipes with extreme proportions) which encompasses all of the non-boundary points.

No triplet of boundary points will be collinear, i.e. lying on a single line. For example, for sample input 1, you are guaranteed not to encounter points at (3, 3), (4, 3), or (5, 3) since they would each be collinear with the points (2, 3) and (6, 3).

Input The first line contains an integer N, $3 \leq N \leq 10,000$ representing the number of points. Each of the next N lines contain 2 integers representing the (x, y) coordinates of each point; $0 \leq x, y < 1,000$.

Output The first line of the output should contain a single integer h which is the number of points on the boundary. The next h lines should each contain two

integers representing the (x, y) coordinates of the points. The points should appear in the order which they will be connected but may start at any of the boundary points. There is no need to repeat the final connection back to the first point.

Sample Input 1

```
18
5 9
4 4
6 6
5 6
1 4
7 5
2 3
6 8
6 4
6 5
5 5
6 7
4 6
7 6
5 7
6 3
4 5
3 4
```

Sample Input 2

```
18
6 4
4 7
5 7
6 6
8 2
7 3
4 5
5 4
7 5
1 6
8 5
6 5
3 5
5 5
7 6
4 4
4 3
5 3
```

Sample Output 1

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

Sample Output 2

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