British Informatics Olympiad

Time allowed: 3 hours

The 2016 British Informatics Olympiad

Instructions

You should write a program for part (a) of each question, and produce written answers to the remaining parts. Programs may be used to help produce the answers to these written questions but are not always necessary.

You may use a calculator and the on-line help that your programming language provides. You should have a pen, some blank paper, and an empty USB stick (or other storage device) on which to save your programs. You must not use any other material such as files on a computer network, books or other written information. You may not communicate with anyone, other than the person invigilating this paper.

Mark the first page of your written answers with your name, age in years and school/college. Number all pages in order if you use more than one sheet. All of your computer programs should display your name and school/college when they are run, and the storage device you use to submit the programs should also show your name and school/college.

For your programs to be marked, the source code must be saved, along with executables if your language includes a compiler; this includes programs used to help answer written questions. You must clearly indicate the name given to each program on your answer sheet(s).

Sample runs are given for parts 1(a), 2(a) and 3(a). **`Bold text`** indicates output from the program, and `normal text` shows data that has been entered. Where multiple items of input appear on the same line they should be separated by a single space. The output format of your programs should follow the 'sample run' examples. Your programs should take less than *1 second* of processing time for each test.

Attempt as many questions as you can. Do not worry if you are unable to finish this paper in the time available. Marks allocated to each part of a question are shown in square brackets next to the questions. Partial solutions (such as programs that only get some of the test cases correct within the time limit, or partly completed written answers) may get partial marks.

Questions can be answered in any order, and you may answer the written questions without attempting the programming parts.

Hints

- If you can only see how to solve part of a problem it is worth writing a program that solves that part. We want to give you marks and questions are evaluated using multiple tests of differing difficulty. ***Remember, partial solutions may get partial marks.***
- Question 2 is an implementation challenge and question 3 is a problem solving challenge.
- Most written questions can be solved by hand without solving the programming parts.
- Do not forget to indicate the name given to your programs on your answer sheet(s).

Question 1: *Promenade Fractions*

A *promenade* is a way of uniquely representing a fraction by a succession of "left or right" choices. As successive choices are made the value of the promenade changes by combining the values of the promenade before the most recent left choice with the value before the most recent right choice.

If the value before the most recent left choice was *l/m* and the value before the most recent right choice was *r/s* then the new value will be (*l*+*r*) / (*m*+*s*). If there has never been a left choice we use *l*=1 and *m*=0; if there has never been a right choice we use *r*=0 and *s*=1.

Fractions are **always** used in their lowest form; recall that *a/b* is in its lowest form if it is not possible to divide *a* and *b* by a common factor. Values generated by the formula will automatically be in their lowest form. Fractions are allowed to have *a* larger than *b*.

We will write our promenades as a sequence of Ls (for left choices) and Rs (for right choices).

For example, to form the promenade LRLL (using ∅ to represent the promenade before any choices are made):
- The value of ∅ is (1+0)/(0+1) = 1/1;
- The value of L is 1/2. Before the most recent left choice we bhad ∅ (= 1/1). There has not yet been a right choice, so we use *r*=0 and *s*=1. So the value of L is (1+0)/(1+1) = 1/2;
- LR = 2/3 as we use the values of ∅ (before the left choice) and L (before the right choice);
- LRL = 3/5 as we use the values of LR and L;
- LRLL = 4/7 as we use the values of LRL and L.

1(a) [23 marks]

Write a program which reads in a promenade of between 1 and 10 (inclusive) uppercase letters (each `L` or `R`).

You should output the promenade's value as a fraction in its lowest form.

Sample run 1

```
LRLL
4 / 7
```

Sample run 2

```
RL
3 / 2
```

1(b) [2 marks]

What is LRL + LLLL as a promenade?

1(c) [3 marks]

How many Ls and how many Rs does the promenade representing 1 / 1,000,000 contain?

1(d) [3 marks]

Does any promenade represent a negative fraction? Justify your answer.

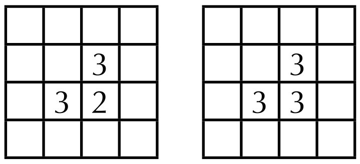
Question 2: *Migration*

Your task in this question is to model people moving across a virtual landscape, consisting of an infinite grid of squares.

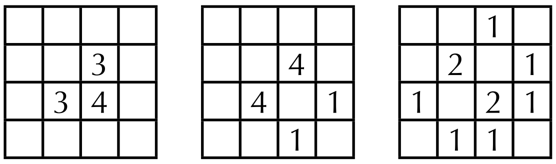
If a square contains 4 or more people it is *overcrowded*. If a square becomes overcrowded then people will *migrate* from the square to the neighbouring squares; these are the four squares directly adjacent horizontally and vertically. A migration consists of 4 people from a square moving simultaneously, one to each neighbouring square.

On each *step* of the simulation a new person is added to the landscape. Whilst there are any overcrowded squares, one of them (it does not matter which) will migrate; this is repeated until there are no more overcrowded squares. This ends this step of the simulation.

For example, suppose that the landscape starts with the only people as shown in the left figure. If the next step is to add a person to the square currently containing 2 people, there will be no overcrowding and the step will end with the grid as shown in the right figure.



If, on the next step, another person is added to the same square we will have an overcrowded square (left figure) and migration will take place (middle figure). This causes two overcrowded squares and these will successively migrate (right figure). As there are now no more overcrowded squares, the step will end.



The following method will be used to add people to the grid:

- We will place people (at the start of steps) within a 5×5 section of the landscape. The top row of this section contains positions 1 to 5 (from left to right), the next row positions 5 to 10, etc... so that the bottom right corner is position 25.
- You will be given the position of the square that receives a person in the first step.
- You will be given a sequence of numbers (each between 0 and 24). These will be used in order. When the last number has been used, return to the beginning of the sequence.
- On each successive step, generate the position for the next person by adding the next number in the sequence to the position used in the previous step. If this number is greater than 25, take 25 off total so that you are left with a position from 1 to 25.

For example, if the starting position is 10 and the sequence is 5 then 6, people will be placed at positions 10, 15, 21, 1, 7, 12, 18, 23, 4, etc...

NB: You must still model the landscape outside of this 5×5 section.
Migrations can still cause movements out of and into this section.

2(a) [24 marks]

Write a program that models migrations across an initially empty virtual landscape.

Your program should first read three integers: the starting position p ($1 \leq p \leq 25$) then a sequence size s ($1 \leq s \leq 6$) and finally n ($1 \leq n \leq 1000$). You should then read in s integers (each between 0 and 24 inclusive) indicating the sequence values in order.

You should output the 5×5 section of the landscape as it appears after n steps of the simulation. You should indicate the number of people in each square; do not indicate the position values.

Sample run

```
8 1 6
0

0 0 1 0 0
0 1 2 1 0
0 0 1 0 0
0 0 0 0 0
0 0 0 0 0
```

2(b) [2 marks]

Consider an empty virtual landscape. What is the smallest number of people that can be added to a 5×5 section of the landscape such that a migration occurs from outside to inside the section?

2(c) [4 marks]

The following grid has been generated with 8 steps and a sequence size of 3. Give a corresponding input that produces this output. How many such inputs are there?

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

2(d) [4 marks]

If you are given the landscape at the end of a step, along with the position where the person was added at the start of the step, can you always determine the landscape at the start of the step? Justify your answer.

Question 3: *Prime Connections*

A *prime number* is a whole number, greater than 1, that can only be divided by itself and the number 1. Two prime numbers are *connected* if the difference between them is 2^n for some whole number $n \geq 0$; e.g. possible differences are 1, 2, 4, 8, 16, 32, …

A *path* is a sequence of (at least two) prime numbers, without repetition, where adjacent numbers in the sequence are *connected*. If the first number in the sequence is p and the last number is q then we say the path is *between* p and q.

The *length* of a path is the total number of prime numbers used. There may be multiple paths between two prime numbers; the lengths of these paths may be different.

For example:
- 13 is connected to 5 ($13 - 5 = 8 = 2^3$), 5 is connected to 3 ($5 - 3 = 2 = 2^1$) and 3 is connected to 2 ($3 - 2 = 1 = 2^0$);
- As 13 and 5 are connected there is a path between them (13—5) whose length is 2;
- There is a path from 13 to 2 (13—5—3—2) whose length is 4;
- There is a longer path from 13 to 2 (13—17—19—3—2) whose length is 5.

You will be given an upper limit on the primes you are allowed to use. For example, if the limit was 18 then the path 13—17—19—3—2 would *not* be permitted as it includes a prime above this limit.

3(a) [27 marks]

Write a program to determine the length of the *shortest* path between two primes.

Your program should input three integers in order: l ($4 \leq l \leq 2^{24}$) indicating the highest value you are allowed to use, followed by the primes p then q ($2 \leq p < q < l$). You will only be given input where there is a path between p and q using values below l.

You should output the length of the shortest path.

Sample run

```
100 2 13
4
```

3(b) [2 marks]

How many different *paths* are there between 2 and 19 with a upper limit of 20?

3(c) [3 marks]

How many pairs of *connected* primes are there with an upper limit of 250,000? (Reversing the order of the primes does *not* count as a different pair.)

3(d) [3 marks]

Suppose that there are two (different) paths of length n between p and q, and that both of these paths contain exactly the same primes. What can you say about the length of the shortest path between p and q?

Total Marks: 100

End of BIO 2016 Round One paper