British Informatics Olympiad

Time allowed: 3 hours

The 2020 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: *Roman Look-and-Say*

The *Roman look-and-say* description of a string (of `I`s, `V`s, `X`s, `L`s, `C`s, `D`s and `M`s) is made by taking each block of adjacent identical letters and replacing it with the number of occurrences of that letter, given in *Roman numerals* **(*)**, followed by the letter itself. A block of adjacent identical letters is never broken into smaller pieces before describing it.

For example:

- `MMXX` is described as "two `M`s followed by two `X`s". Since two is `II` in Roman numerals, this is written as `IIMIIX`;
- `IIMIIX` is described as `IIIIMIIIIX`, which is "two `I`s, one `M`, two `I`s, one `X`";
- `IIIIMIIIIX` is described as `IVIIMIVIIX`;
- It is *not* valid to describe `III` as, "two `I`s, one `I`" `IIIII`.

Note that Roman look-and-say descriptions are *not* necessarily Roman numerals.

1(a) [25 marks]

Write a program that reads in a Roman numeral representing a number between 1 and 3999 inclusive, followed by n ($1 \leq n \leq 50$).

You should apply the *Roman look-and-say* description n times and then output the number of `I`s in the final description, followed by the number of `V`s.

Sample run 1

```
MMXX 1
4 0
```

Sample run 2

```
MMXX 3
6 2
```

1(b) [2 marks]

How many Roman numerals (from 1 to 3999 inclusive) have a Roman look-and-say description that is also a Roman numeral? List these *descriptions*.

1(c) [4 marks]

The Roman look-and-say descriptions are generated for all the Roman numerals from 1 to 3999 (inclusive). How many distinct descriptions are there?

(*) *Roman numerals* are conventionally defined to represent numbers using seven letters: `I`=1, `V`=5, `X`=10, `L`=50, `C`=100, `D`=500 and `M`=1000. Numbers other than these are formed by placing letters together, from left to right, in descending order of size, and adding their values. The basic rule is to always use the biggest numeral possible (e.g. 15 is represented as `XV` but never as `VVV`, `VX` or `XIIIII`).

Letters may not appear more than three times in a row, so there are six exceptions to these rules – the combinations `IV`, `IX`, `XL`, `XC`, `CD` and `CM`. In these cases a letter is placed before one of greater value and the smaller value is subtracted from the larger. E.g. `CD` = 400. These are the *only* exceptions so, for example, `MIM` is not valid.

Question 2: *Alpha Complex*

Alpha Complex consists of many connected rooms. Each room is identified by a single (unique) letter and each exit from a room is marked with the letter identifying the room to which it is connected.

A *spy* is moving systematically through Alpha Complex. For each room they keep track of whether they have visited it an even or an odd number of times. They also record, for each room and for each exit, whether they have left the room through that exit an odd or even number of times.

The spy moves between rooms according to the following rules:

- If they have visited the room an odd number of times, they will leave through the exit which is marked with the first letter alphabetically;
- If they have visited the room an even number of times, they will find the first exit alphabetically that they have left through an odd number of times. If that is the last exit alphabetically in this room they will leave through it, otherwise they will leave through the next exit alphabetically.

When the spy starts exploring Alpha Complex, they have visited their starting room an odd number of times (i.e. once), each other room an even number of times (i.e. zero) and have left through each exit an even number of times (i.e. zero).

For example, suppose that they start in room `X`, which has exits to `B`, `I` and `O`:

- They have visited `X` an odd number of times, so they leave the room through the first exit alphabetically, which is `B`;
- When they are next in X they will have visited it an even number of times. The first exit alphabetically that they have left through an odd number of times is `B`. As this is not the last exit alphabetically, they leave through exit `I`;
- Subsequent visits to the room will see them leave through `B`, `O`, `B`, `I`, `B`, `O`, …

Alpha Complex consists of r ($r \geq 3$) rooms, identified by the first r letters of the alphabet. The spy is in possession of a secret *plan*, giving the connections between the rooms. This plan is an ordered list of $r-2$ letters and the spy can construct a map of the complex as follows:

- The spy will *choose* the first room alphabetically which has not yet been chosen and which is not in the plan. The chosen room is connected to the first room in the plan. The first room is then removed from the plan;
- The above step is repeated until the plan is empty;
- There will be two rooms which have not yet been chosen. These two rooms are connected together.

If two rooms are connected, each room has an exit to the other room.

For example, suppose Alpha Complex has 6 rooms and the plan is `E`, `A`, `E`, `D`:

- The first room which is not in the plan is `B`, which gets connected to `E` (the first room in the plan). The updated plan becomes `A`, `E`, `D`. Note that only `B` has been chosen and that the other instance of `E` in the plan is not removed;
- The next chosen room will be `C`, which gets connected to `A`;
- The next chosen room will be `A` (it has not been previously chosen and is no longer in the plan) and it is connected to `E`;
- `E` is now chosen and connected to `D`;
- The plan is now empty. As `D` and `F` were never chosen, they are now connected.

2(a) [24 marks]

Write a program that tracks the spy through Alpha Complex.

Your program should first input a string consisting of n ($1 \le n \le 8$) uppercase letters (each from the first n+2 letters of the alphabet) indicating the spy's plan of Alpha Complex, followed by p and q ($1 \le p < q \le 1{,}000{,}000$) indicating moves the spy will make.

The spy starts in room `A`.

You should first output n+2 lines. The first containing (in alphabetical order) the rooms connected to `A`, the second those connected to `B`, etc. These should be followed by a line containing the spy's location after p and q moves.

Sample run

```
EAED 1 2
CE
E
A
EF
ABD
D
CA
```

2(b) [2 marks]

What is the plan if Alpha Complex has 3 rooms and the connected rooms are `A-B` and `A-C`? What is the plan if there are 6 rooms and the connections are `A-B`, `A-C`, `A-D`, `A-E` and `A-F`?

2(c) [4 marks]

After an unknown number of moves in Alpha Complex the spy arrives in a room having forgotten if it has been visited an odd or an even number of times. How can they continue their systematic exploration without deviating from their original intended route? Justify your answer.

2(d) [4 marks]

If Alpha Complex contains 8 rooms, how many different plans are there where the first four connections the spy adds when constructing the map are (in some order) `A-E`, `B-F`, `C-G` and `D-H`?

Question 3: *False Plan*

A spy, currently working their way through an enemy compound, has been given a *false plan*. The plan is an ordered list of letters. In order to make the plan look realistic, the number of adjacent identical letters in the plan has been limited.

For example, suppose that the plan contains four letters, each of which is an `A` or a `B`, and that there are never more than two adjacent identical letters. There are 10 possible plans:

```
AABA
AABB
ABAA
ABAB
ABBA
BAAB
BABA
BABB
BBAA
BBAB
```

These have been listed in *alphabetical* order.

3(a) [27 marks]

Write a program to determine the n^{th} possible plan.

Your program should input a line containing three integers p, q, r ($1 \le p, q, r \le 12$) indicating (in order) that the first p letters of the alphabet can be used, no more than q adjacent identical letters are permitted and that the plan should contain exactly r letters. You should then input a line containing a single number n ($1 \le n < 2^{63}$).

You will only be given input where n is no greater than the number of possible plans.

You should output the n^{th} possible plan.

Sample run

```
2 2 4
7
BABA
```

3(b) [3 marks]

Suppose `A`s, `B`s, `C`s and `D`s are permitted and there are never more than two adjacent identical letters. Consider the list of possible three letter plans; which plan is `CCA` within this list? How about (for eight letter plans) `CCABABCC`?

3(c) [5 marks]

Suppose there is a plan that is in the n^{th} position both when the plans are ordered alphabetically and when they are ordered reverse alphabetically. What can you determine about p, q, and r? Justify your answer.

Total Marks: 100 End of BIO 2020 Round One paper

British Informatics Olympiad

Time allowed: 3 hours

The 2019 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: *Palindromic Numbers*

A *palindromic number* is one that is the same when its digits are reversed.

For example:
- 98789 is a palindrome;
- 12344321 is a palindrome;
- 12345 is not a palindrome as it becomes 54321 when reversed.

Except for 0, a palindromic number's leftmost digit must be non-zero.

1(a) [25 marks]

Write a program that reads in a positive integer of up to 20 digits.

You should output the smallest palindromic number that is higher than the input.

Sample run 1

```
17
22
```

Sample run 2

```
343
353
```

1(b) [2 marks]

What is the largest difference between a palindromic number (of up to 20 digits) and the next highest palindromic number?

1(c) [4 marks]

How many integers are there, between 1 and 99999 inclusive, that are *not* the sum of two palindromic numbers?

Question 2: *Trail*

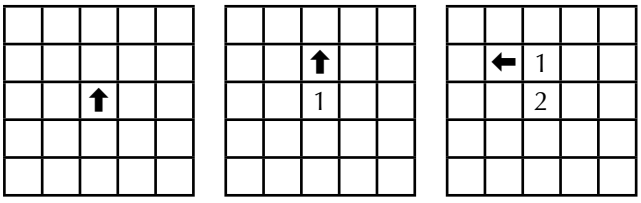
An *explorer* is moving across a rectangular grid of squares, leaving behind a *trail*. Keen to experience the unknown, they only move into squares that do not contain part of the trail. The trail fades over time and the explorer can re-enter a square once its trail has disappeared, although they will leave behind a new trail.

If the trail fades after *n* moves, it means that the trail in a square will disappear once the explorer has finished moving *n* times.

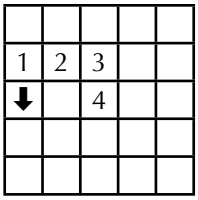
The explorer is following a set of instructions. If the instructions indicate *left* or *right* they will turn 90 degrees in the corresponding direction, otherwise they will stay facing *forward*. The explorer will then move to the adjacent square in the direction they are facing, so long as it does not contain the trail. If this is not possible, they will turn 90 degrees to the *right* and try to move. This will be repeated until they have moved or tried all four directions.

Each entry in the explorer's instructions is either `L` (*left*), `R` (*right*) or `F` (*forward*). The explorer works through the entries in order, restarting at the beginning once they have all been used.

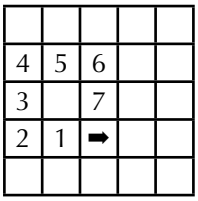
For example, suppose the trail fades after the explorer has moved 8 times and their instructions are `FL`. (In the diagrams, the arrow will indicate the explorer and the direction they are facing, with the numbers indicating the age of the trail.)



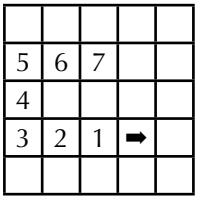
The explorer first moves forward, and then moves to their left. After this second move they are facing in a new direction and the trail covers two squares.



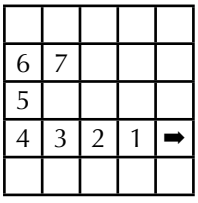
The explorer will now restart their instructions. Another move forward in the direction they are facing, followed by a move to the left.



After another three moves the explorer's next instruction is a *left*. They turn to the left but are facing a square that contains the oldest part of the trail. They therefore turn to the right …



… and move into the empty square. They have now moved 8 times since the oldest part of the trail was formed, so it disappears.



They continue with the next instruction, which is *forward*.

2(a) [24 marks]

Sample run

```
8 FL 9
(2,-1)
```

Write a program that tracks the explorer.

Your program should first input an integer t ($1 \leq t \leq 100$) indicating the number of moves for the trail to disappear, followed by a word i of between 1 and 10 upper case letters (each `L`, `R` or `F`) indicating the explorer's instructions, followed by an integer m ($1 \leq m \leq 10000$) indicating how many moves the explorer makes.

If the explorer is ever unable to make a move they will stop and not attempt to make any more moves.

You should output the co-ordinates of the explorer after making their moves.

The explorer starts at (0,0) facing (0,1). To their left is (-1,0) and to their right is (1,0).

2(b) [2 marks]

The introduction to this question gave the diagram after `8 FL 9`. Give the diagram after `8 FL 16`.

2(c) [3 marks]

If the trail never disappears and the explorer's instructions are `L`, how many moves are required for every square at (x,y) to be visited, where $-10 \leq x \leq 10$ and $-10 \leq y \leq 10$?

2(d) [5 marks]

The explorer's instructions are `LLRFFF` and they are able to move indefinitely. What is the largest possible number of moves before the trail disappears?

Question 3: *Block-chain*

A set of children's blocks, each illustrated with a single different letter, have been chained together in a line. They have been arranged so that it is not possible to find three (not necessarily adjacent) letters, from left to right, that are in alphabetical order.

For example, if there are four blocks (A, B, C and D) the possible block-chains are:

```
ADCB    BADC    BDAC    BDCA    CADB
CBAD    CBDA    CDAB    CDBA    DACB
DBAC    DBCA    DCAB    DCBA
```

3(a) [24 marks]

Write a program that enumerates block-chains.

Your program should input a single integer l ($1 \leq l \leq 19$) indicating that the blocks are illustrated with the first l letters of the alphabet, followed by a word p of between 1 and l uppercase letters indicating (in order) the leftmost letters of the block chain. p will only contain letters take from the first l letters of the alphabet and will not contain any duplicates.

You should output a single integer giving the number of possible block-chains that begin with p.

Sample run

```
4 CB
2
```

3(b) [2 marks]

List the valid block-chains containing the blocks `B`, `I` and `O`.

3(c) [4 marks]

A block-chain containing the first n ($1 \leq n \leq 13$) letters of the alphabet is attached to the left of a block-chain containing the last m ($1 \leq m \leq 13$) letters of the alphabet, forming a new block-chain. What were the original block-chains? Justify your answer.

3(d) [5 marks]

Suppose all the valid block-chains containing the first 19 letters of the alphabet are sorted into alphabetical order. Which one comes first? Which one comes 1,000,000,000th?

Total Marks: 100

End of BIO 2019 Round One paper

British Informatics Olympiad

Time allowed: 3 hours

The 2018 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: *Debt Repayment*

At the end of every month interest is added to a *debt* (initially 100) and then repayments are taken off. The *interest* is a fixed percentage of the current debt. The *repayment* is also a fixed percentage of the debt (after the interest has been added) or 50, whichever is larger. If the repayment is greater than the debt it is reduced to match the debt. The cycle of interest and repayment is continued until the debt has been reduced to 0 and paid off.

For example, suppose the interest is 10% and repayments are 50%:

- At the end of the first month the interest is 10, increasing the debt to 110;
- The repayment is 55 (50% of 110) leaving the debt at 55;
- At the end of the second month the interest is 5.5, increasing the debt to 60.5;
- The repayment is 50 (as 50% of 60.5 is less than the minimum amount), leaving the debt at 10.5;
- At the end of the third month the interest is 1.05, increasing the debt to 11.55;
- The repayment is 11.55 (the minimum payment of 50 is reduced to match the debt) and the debt is paid off.
- The total amount repaid on the debt is 116.55

When calculating percentages the amount is rounded *up* to 2 decimal places. E.g. 10.201 and 10.207 would both be rounded up to 10.21.

1(a) [26 marks]

Write a program that reads in the interest percentage followed by the repayment percentage. Percentages will be integers between 0 and 100 (inclusive).

You should output the total amount repaid.

You will always be given input that allows the debt to be paid off.

Sample run 1

```
10 50
116.55
```

1(b) [2 marks]

Given an interest percentage of 43% and a repayment percentage of 46%, how many payments will be made to pay off the debt?

1(c) [3 marks]

Which interest and repayment percentages (integers between 0 and 100) lead to the largest amount repaid on a debt that is paid off?

Question 2: *Decoder Ring*

A *decoder ring* consists of two adjacent dials and is used to encrypt (or decrypt) messages. The two dials are lined up, so that each position on the first is touching one on the second, and dials can rotate so that they can be aligned in different ways. The first dial has 26 positions, lettered from A to Z in order, and the second dial has the same letters but not necessarily in the same order. A letter is encrypted by finding it on the first dial and using the corresponding touching letter on the second dial.

For example, suppose that the second dial has been lettered from Z to A (i.e. reverse order) and that the A on the first dial is touching Z on the second:

- The letter A would be encrypted to the letter Z, the letter B by the letter Y etc.
- If the second dial is now rotated until the A on the first dial is touching X on the second, the letter A would be encrypted to the letter X, the letter B by W etc.

The order of the letters on the second dial will be generated as follows, from the number n:

- Place the letters A to Z clockwise in order around a circle;
- Starting with A count clockwise round the circle until n is reached;
- *Remove* the selected letter from the circle — it becomes the first letter on the second dial;
- Starting from where you left off, count to n again to select the next letter;
- Continue until all the letters have been selected.

For example, if n is 5 the letters will be chosen in the order: E, J, O, T, Y, D, K, Q, W, ...

The dials will be aligned so that the first letter selected for the second dial is initially touching the letter A on the first dial, the second letter selected touching B etc. After each rotation of the dials, the letter on the second dial previously touching B on the first dial will be touching A on the first dial, etc.

A word is encoded by encrypting each letter in turn, after each encryption rotating the dial by a single position.

For example, if n is 5 the word ABCD will be encrypted as EOYK.

2(a) [24 marks]

Write a program that encrypts a word.

Your program should first input a single integer n ($1 \leq n \leq 1000000$) followed by a word w of between 1 and 8 upper case letters.

You should output the first 6 letters of the second dial generated from n, followed by the encrypted version of w.

Sample run

```
5 ABCD
EJOTYD
EOYK
```

2(b) [3 marks]

What are the first 6 letters of the second dial generated from n = 1,000,000,000?

2(c) [4 marks]

The word ABCDEFGHIJKLMNOPQRSTUVWXYZ is to be encrypted. Does a second dial exist that will encrypt this to a word that contains all 26 letters? Justify your answer.

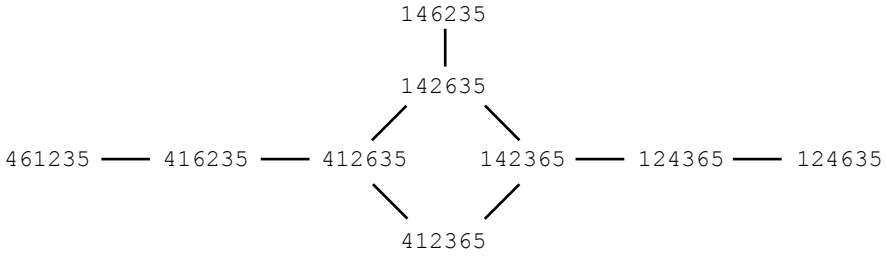
2(d) [4 marks]

A word is encrypted *without rotating the dials* and this encrypted word encrypted again and again until the first time the original word is produced. This will always occur eventually. What is the largest number of encryptions that might be required?

Question 3: *Serial Numbers*

A *serial number* can be transformed into another, by swapping two adjacent digits, so long as one of those two digits is adjacent to a digit whose value lies between them. Two *serial numbers* will be called *equivalent* if there is a sequence of transformations which changes one into the other, and the *distance* between them is the *minimum* number of such transformations required.

For example, the following shows all the serial numbers that are equivalent to `146235`, the lines showing all the valid transformations:



The distance from `461235` to `124635` is 6, and from `412635` to `142635` is 1.

3(a) [24 marks]

Write a program that determines the largest distance between a *serial number* and any equivalent number.

Your program should input a single integer d ($1 \leq d \leq 9$) indicating the number of digits in the serial number, followed by a d digit serial number which will consist of the numbers 1,...,d without repetition.

You should output a single integer giving the largest distance.

Sample run

```
6
461235
6
```

3(b) [4 marks]

What is the largest distance between *any* two serial numbers equivalent to 326451? How about 183654792?

3(c) [6 marks]

What is the size of the largest set of serial numbers, none of which are equivalent, each consisting of the digits 1,...,5 without repetition? What is the size if the serial numbers consist of the digits 1,...,9 without repetition?

Total Marks: 100

End of BIO 2018 Round One paper

British Informatics Olympiad

Time allowed: 3 hours

The 2017 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: *Coloured Triangles*

A *coloured triangle* is created from a row of squares, each of which is *red*, *green* or *blue*. Successive rows, each containing one fewer square than the last, are generated by considering the two touching squares in the previous row. If these squares are identical, the same colour is used in the new row. If they are different, the missing colour is used in the new row. This is continued until the final row, with only a single square, is generated.

In the following three example triangles `R`, `G` and `B` have been used to represent the colours. Note that each row is generated from the row above.

```
G G
 G
```

```
R B
 G
```

```
R R G B R G B B
 R B R G B R B
  G G B R G G
   G R G B G
    B B R R
     B G R
      R B
       G
```

1(a) [23 marks]

Write a program which reads in a starting row of between 1 and 10 (inclusive) uppercase letters (each `R`, `G` or `B`).

You should output the colour (`R`, `G` or `B`) of the square in the final row of the triangle.

Sample run 1

```
RG
B
```

Sample run 2

```
RBRGBRB
G
```

1(b) [3 marks]

How many possible rows of nine squares are there that generate `RRGBRGBB`? List them.

1(c) [4 marks]

Suppose you have a triangle where, on each row, only the colour of a single square is known. How many ways can the unknown squares be coloured so that the triangle is consistent with the rules for generating a *coloured triangle*? Justify your answer.

1(d) [3 marks]

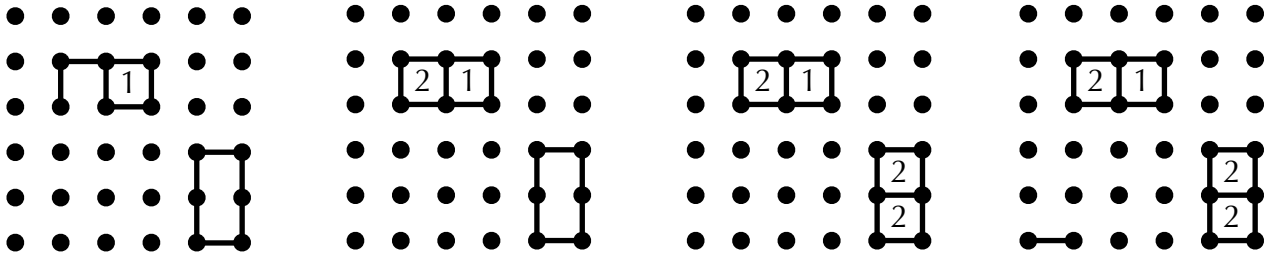
If the first row contains 4 squares, the colour of the square in the final row only depends on the extreme left and right of the first row. This is not true if, for example, the first row contains 5 squares. Find a larger size of row which shares this unusual property.

Question 2: *Dots and Boxes*

The game of *dots and boxes* is being played on a 6×6 grid of dots. Two players take turns selecting two adjacent dots (horizontally or vertically) and joining them by an edge. If the edge finishes off any (1×1) squares then those squares are won by the current player who then gets another turn, otherwise it becomes the other player's turn. Two adjacent dots can only be joined once.

For convenience in this question, we will number the dots 1 to 36, with the top row running from 1 (left) to 6 (right), the next row from 7 to 12 etc.

For example, the following sequence of grids shows a game in progress:



We start with the grid on the left where player 1 has won a single square. It is player 2's turn and they choose to join dots 14 and 15, finishing off the square towards the top left. Player 2 now gets another turn and chooses 29—30 which simultaneously finishes off both bottom right squares. Player 2 now gets another turn and joins 31—32 on the bottom left. It is now player 1's turn again.

In this question you will simulate two players who follow a very simple strategy.

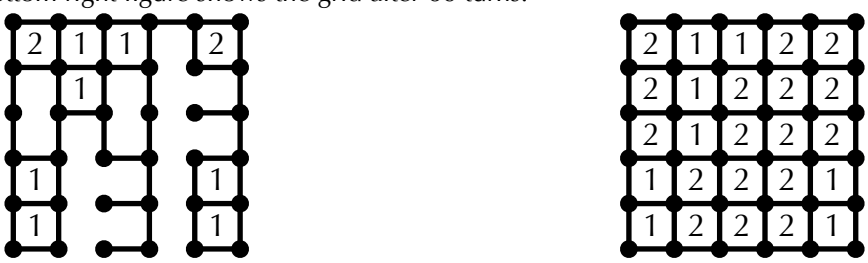
Each player uses a *modifier* and keeps track of a *position*. At the start of their turn, a player increases their position by their modifier and looks at the dot in the new position. If it is possible they then join an edge to that dot. If it is not possible they repeatedly increase the position by 1 (rather than the modifier) until they find a dot to which they can connect an edge.

A player increasing their position past dot 36 starts again at dot 1. E.g. If a position of 35 is modified by 3, the new position will be 2.

When trying to join an edge from a dot each player first looks to see if an edge can be added going upwards. If this is not possible, player 1 tries edges in a clockwise direction (i.e. first seeing if an edge goes to the right, then downwards and finally to the left) and player 2 tries edges in a counterclockwise direction.

For example, suppose no dots have been joined, player 1 starts with position 4 and a modifier of 10, and player 2 starts with position 14 and a modifier of 23:

- Player 1 increases their position to 14. An edge is added upwards joining 14—8.
- Player 2 increases their position to 1 (14 + 23 = 37, which is a single position beyond 36). An edge cannot be added upwards or to left (dot 1 is at the upper left corner), so an edge is added downwards joining dot 1 to dot 7.
- Player 1 increases their position to 24. An edge is added upwards joining 24—18.
- Player 2 increases their position to 24. An edge already exists upwards, so an edge is added to the left joining 24—23.
- The bottom left figure shows the grid after 46 turns; it is player 1's turn and they are at position 8. They look at position 18 (no possible edges), then 19 (no possible edges), then 20 and join 20—21.
- The bottom right figure shows the grid after 60 turns.



2(a) [24 marks]

Write a program that plays a game of *dots and boxes*.

Your program should first read five integers: the starting position p_1 ($1 \le p_1 \le 36$) then modifier m_1 ($1 \le m_1 \le 35$) for player 1, followed by starting position p_2 ($1 \le p_2 \le 36$) then modifier m_2 ($1 \le m_2 \le 35$) for player 2, followed by the number of turns to simulate t ($1 \le t \le 60$).

You should output a grid showing which squares have been won after t turns. Output an `X` for player 1, an `O` for player 2, and a `*` for a square that has not yet been won. This should be followed by the number of squares won by the first player, then the number of squares won by the second player.

Sample run

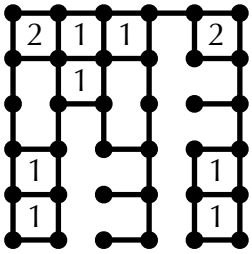
```
4 10 14 23 47

O X X * O
* X * * *
* X * * *
X * * * X
X * * * X

8 2
```

2(b) [2 marks]

Suppose the grid is in the following configuration and it is player 1's turn. If players can adopt any strategy (not just the very simple strategy), what is the maximum number of squares player 1 can win at the end of the game?



2(c) [3 marks]

How many different sets of input, where t=60, lead to player 2 winning all 25 squares?

2(d) [5 marks]

When an edge is added either 0, 1 or 2 squares are completed. Suppose the players are allowed to use any strategy, and the game is played until every edge has been added. If 2 squares have never been simultaneously completed during the game, what can you say about the pair of dots that were joined on the last turn? Justify your answer.

Question 3: *Mystery Parcel*

A shop is running a promotion and giving away *mystery parcels*, containing one or more items. As these are promotional, items are not very exciting (items that weigh the same are indistinguishable) and the total weight of items in each parcel is the same. Parcels containing the same combination of items (order does not matter) are themselves indistinguishable.

For advertising purposes the shop wishes to calculate the number of ways it can distribute the parcels.

For example, suppose there are 2 parcels containing a total of 4 items, each item weighing 1, 2 or 3 units. There are 10 different ways the parcels can be distributed. They might be constructed in 8 ways:

```
 1 1     —    1 1
 1 2     —    1 2
 2 2     —    2 2
 1 3     —    1 3
 2 3     —    2 3
 3 3     —    3 3
 3 1     —    2 2
1 1 1    —     3
```

In each of the first 6 pairings the parcels are indistinguishable, so there is only a single way in which they can be distributed. In each of the last 2 pairings the parcels can be distinguished, so they could each be distributed in 2 different ways; i.e. it can be distinguished which is distributed first.

3(a) [25 marks]

Write a program to determine the number of ways parcels can be distributed.

Your program should input four integers in order: p ($1 \le p \le 5$) indicating the number of parcels, i ($1 \le i \le 10$) indicating that items can weigh any integer from 1 to i inclusive, n ($1 \le n \le 25$) indicating the total number of items in all the parcels, and w ($1 \le w \le 25$) indicating the weight of each parcel.

You should output the number of ways parcels can be distributed. You will not be given input that requires an answer greater than 2^{31}.

Sample run

```
2 3 4 3
3
```

Marks are available for the case where $p = 1$

3(b) [2 marks]

How many ways can 3 parcels, containing a total of 6 items weighing 1, 2 or 3 units, be distributed?

3(c) [6 marks]

There are 92378 distinguishable parcels containing 10 items (when $i = 10$) and the shop has created an index of these parcels. Given two parcels, the one with the largest number of weight-10 items appears earliest in the index. If those are equal, the order depends on the number of weight-9 items, and so on. E.g. `8888855555` is before `8888855554` which is before `8888777777`.

What is the contents of the parcel at position 50,000? What is the position of the parcel containing one item of each weight from 1 to 10?

Total Marks: 100 End of BIO 2017 Round One paper

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.

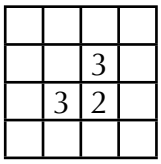
Question 2: *Migration*

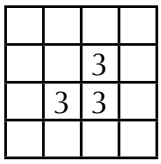
Your task in this question is to model people moving across a virtual landscape, consisting of an infinite grid of squares.

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.

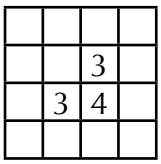
On each *step* of the simulation a new person is added to the landscape. Whilst there are any overcrowded squares, one of them (it does not matter which) will migrate; this is repeated until there are no more overcrowded squares. This ends this step of the simulation.

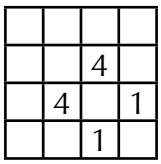
For example, suppose that the landscape starts with the only people as shown in the left figure. If the next step is to add a person to the square currently containing 2 people, there will be no overcrowding and the step will end with the grid as shown in the right figure.

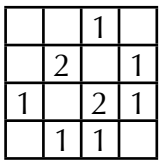




If, on the next step, another person is added to the same square we will have an overcrowded square (left figure) and migration will take place (middle figure). This causes two overcrowded squares and these will successively migrate (right figure). As there are now no more overcrowded squares, the step 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 ($1 \le p \le 25$) then a sequence size s ($1 \le s \le 6$) and finally n ($1 \le n \le 1000$). 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

British Informatics Olympiad

Time allowed: 3 hours

The 2015 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: *Block Palindromes*

A *palindrome* is a word that shows the same sequence of letters when reversed. If a word can have its letters grouped together in two or more blocks (each containing one or more adjacent letters) then it is a *block palindrome* if reversing the order of those blocks results in the same sequence of blocks.

For example, using brackets to indicate blocks, the following are *block palindromes*:

- BONBON can be grouped together as (BON)(BON);
- ONION can be grouped together as (ON)(I)(ON);
- BBACBB can be grouped together as (B)(BACB)(B) or (BB)(AC)(BB) or (B)(B)(AC)(B)(B)
  Note that (BB)(AC)(B)(B) is *not* valid as the reverse (B)(B)(AC)(BB) shows the blocks in a different order.

1(a) [23 marks]

Write a program which reads in a word of between 2 and 10 (inclusive) uppercase letters.

You should output a single number, the number of different ways the input can be grouped to show it is a *block palindrome*.

Sample run

```
BBACBB
3
```

1(b) [2 marks]

Give all the groupings of AABCBAA that show it is a *block palindrome*.

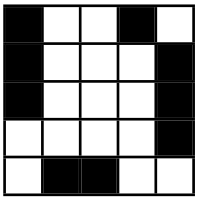
1(c) [6 marks]

Suppose that all the groupings of a *block palindrome* contain an even number of blocks. What can you say about the length of the *block palindrome*? How many different groupings can it have? Justify both your answers.

Question 2: *Battleships*

In the game of *battleships* players secretly position ships of various sizes on a board and then try to determine the position of their opponent's ships. In this question we will consider the placement of one player's ships on a 10×10 board of squares. Each ship is placed either horizontally or vertically, covering an exact number of adjacent squares which are all on the board; a ship of size 1×*n* is called an *n*-ship. No part of any ship can be placed in a square that is adjacent (horizontally, vertically or diagonally) to part of another ship.

For example, part of a board showing four ships (two *3*-ships, a *2*-ship and a *1*-ship) is shown to the right. This is ***not*** valid as the *1*-ship is touching one of the *3*-ships diagonally. If the 1-ship was moved one square to the left everything shown would be valid.



The bottom left corner of the board has co-ordinate (0, 0). Co-ordinates are given as (x,y).

Our player starts by choosing (non-negative integer) values for `a`, `c`, `m` and `r` and then places each ship in turn using the following algorithm:

We are using the notation X←Y to mean "set X to Y"
X *mod* Y is equivalent to the remainder when X is divided by Y

1. `r` ← (`a` × `r` + `c`) *mod* `m`
2. Use the units digit of `r` as an *x* co-ordinate and the tens digit of `r` as a *y* co-ordinate
3. `r` ← (`a` × `r` + `c`) *mod* `m`
4. Starting at the calculated co-ordinates and *only* if it is valid, if `r` is even place the ship going horizontally to the right or if `r` is odd place it vertically upwards
5. If the ship is placed stop, otherwise continue from step 1.

Step 3 is never skipped, even if the value of `r` does not affect the validity of the ship in step 4.

For example, suppose the board contains a single *1*-ship at co-ordinate (3,0), that `a`, `c`, `m` and `r` are currently 3, 5, 53 and 20 respectively, and that the player is trying to place a *2*-ship on the board.

First `r` is set to 12 (3×20+5 = 65 and 65 *mod* 53 = 12) which represents co-ordinate (2,1), then `r` is set to 41 indicating that the ship is placed vertically upwards from (2,1) if valid. This is *not* valid as the ship would occupy (2,1) and (2,2) and the first square is diagonally adjacent to (3,0). `r` is then set to 22 representing co-ordinates (2,2), then set to 18 indicating that the ship is placed horizontally in squares (2,2) and (3,2) which is valid.

It is possible, for some values, that this algorithm will never find a valid position for the ship.

2(a) [27 marks]

Write a program that places ships in a game of battleships.

Your program should first read in three integers: a ($1 \leq a \leq 2^{15}$) then c ($1 \leq c \leq 2^{15}$) and finally m ($1 \leq m \leq 2^{15}$); the initial value of r (which is not input) is always 0. You should then follow the player's algorithm to place a *4*-ship, two *3*-ships, three *2*-ships and four *1*-ships (in that order) on an initially empty board.

You will only be given input that allows all the ships to be placed on the board.

You should output ten lines, the i^{th} of which containing information for the i^{th} placed ship. Each line should contain the x then y starting co-ordinates for its ship, followed by an `H` or a `V` indicating whether the ship is placed horizontally or vertically. (For a *1*-ship you should still indicate the appropriate `H` or `V` based on the r value.)

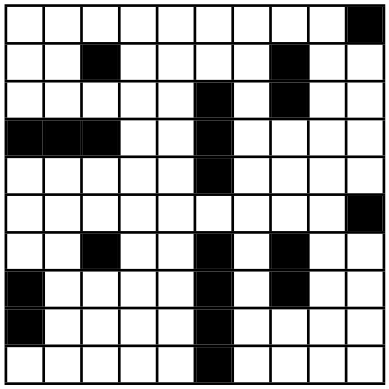
Sample run

```
10 5 9999
```

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

2(b) [2 marks]

If a, c, m and r are set to 2, 3, 17 and 0 respectively, only four co-ordinates will ever be produced by the algorithm in step 2. What are they?



2(c) [3 marks]

The grid to the right corresponds to the sample run.

Another player secretly moves one ship to a valid position. A square can be guaranteed as empty if it is not possible for any ship to occupy it after this move. How many such squares are there?

2(d) [5 marks]

How many valid arrangements are there, on a 5x5 board, of a *4*-ship, a *3*-ship a *2*-ship and a *1*-ship? (All four ships must be on the board.)

Question 3: *Modern Art*

A gallery is displaying pieces of *modern art* by several artists and is considering the different ways of arranging the exhibition. As this is modern art all pieces by the same artist are indistinguishable.

For example, suppose there is a single piece by artist `A`, two by artist `B` and one by artist `C`. There are 12 ways the gallery might arrange the exhibition:

```
ABBC
ABCB
ACBB
BABC
BACB
BBAC
BBCA
BCAB
BCBA
CABB
CBAB
CBBA
```

These have been listed in *alphabetical* order.

3(a) [25 marks]

Write a program to determine the n^{th} way of arranging the exhibition.

Your program should input five integers: *a*, *b*, *c* and *d* (each between 0 and 5 inclusive) indicating the number of works by artists `A`, `B`, `C` and `D` in order, and finally *n* ($1 \leq n \leq 2^{34}$).

You will only be given input where at least one artist is exhibiting a work and *n* is no greater than the number of possible exhibitions.

You should output the string which represents the n^{th} arrangement.

Sample run

```
1 2 1 0 8
BCAB
```

3(b) [2 marks]

If the gallery is exhibiting `AABCCBDD` which arrangement is this?

3(c) [5 marks]

An unspecified number of artists are exhibiting an unspecified number of pieces of modern art. The gallery has exhibited the n^{th} arrangement followed by the $n+1^{st}$ arrangement. For poetic reasons, for each position in the n^{th} arrangement the artist providing the work in the same position in the $n+1^{st}$ arrangement was different. Is it possible that, in the $n+2^{nd}$ arrangement, the artist providing the work in each position is different to that in the $n+1^{st}$ arrangement? Justify your answer.

Total Marks: 100 End of BIO 2015 Round One paper

British Informatics Olympiad

Time allowed: 3 hours

The 2014 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 *2 seconds* 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: *Lucky Numbers*

The *lucky numbers* are produced by taking a list of all the odd numbers and systematically removing some of the numbers. Our first lucky number is 1. We now repeatedly take the next highest number n that is still in the list, mark it as a lucky number and then remove every n^{th} number from the list.

For example, with lucky numbers underlined as we progress:

- Initially we have: 1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25, 27, 29, 31, ...
- The next lucky number is 3, giving us: 1, 3, 7, 9, 13, 15, 19, 21, 25, 27, 31, ...
- Next is 7: 1, 3, 7, 9, 13, 15, 21, 25, 27, 31, ...
- Next is 9: 1, 3, 7, 9, 13, 15, 21, 25, 31, ...

1(a) [25 marks]

Write a program which reads in a single integer between 2 and 10,000 inclusive.

You should output two numbers. Firstly the closest lucky number that is *less* than the input, followed by the closest lucky number that is *greater* than the input.

Sample run

```
5
3 7
```

1(b) [2 marks]

How many numbers less than 100 are lucky?

1(c) [3 marks]

Suppose that you are told the 1,000,000,000th lucky number is X. A program that *correctly* solves 1(a) (on inputs of any size >2) is run on all numbers from 2 to X inclusive. How many *different* results will the program produce?

(NB: A result is a pair of numbers produced by the program; e.g. `3 7` is a single result.)

Question 2: *Loops*

Red and Green are playing a game which consists of placing square tiles on a grid and trying to form *loops* of their own colour. Each tile incorporates a red line and a green line, touching different edges of the tile.

There are six different types of tile that are used in the game. Red and green lines are shown by solid and dashed lines respectively. The numbers by the tiles will be used when inputting grids.

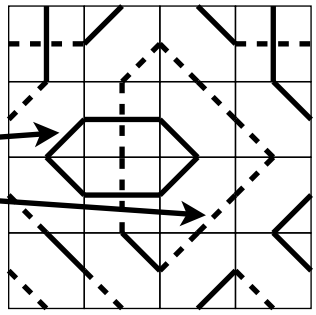
Tiles are placed adjacent to each other on the grid. A line meeting another line of the same colour — across a shared edge between tiles — is treated as a single continuous line across multiple tiles. If the lines are different colours the lines are treated as being separate. A loop is a line that begins on one tile and continues along additional tiles until it rejoins itself on the first tile.

At the end of the game each player scores a single point for each tile that contains part of a line forming a loop of their colour. A tile can contribute towards the score of both players.

In the example to the right there is a single loop; the numbers indicate how the grid will be input.

- This is a red (solid) loop covering six tiles.
- This is not a loop as the green (dashed) line does not rejoin itself.

The red player has scored 6 points and the green player 0.



```
1341
5264
4235
4454
```

2(a) [25 marks]

Write a program that reads in a grid and outputs the score for each player.

Your program should first read in a single integer ($2 \leq n \leq 6$) indicating the size of the (square) grid. You should then read in n lines representing the rows (starting at the top), each of which will contain n digits representing the tiles on the grid in order.

You should output two values: the score for the red played followed by the score for the green player.

Sample run

```
4
1 3 4 1
5 2 6 4
4 2 3 5
4 4 5 4

6 0
```

2(b) [2 marks]

A single tile is changed on the example grid so that Red no longer has more points than Green. How many possible changed grids are there?

2(c) [3 marks]

In how many ways can Red score 16 points on a 4 × 4 grid?

2(d) [5 marks]

Red and Green have just played the game on a 500 × 500 grid. Is it possible for Red to have scored just 1 point more than Green? Justify your answer.

Question 3: *Increasing Passwords*

A password scheme accepts passwords that contain combinations of upper-case letters and digits. The scheme restricts passwords to ones in which letters appear in alphabetical order. The digits 0, ..., 9 are treated as coming after Z in alphabetical order; lower digits are treated as coming before higher digits alphabetically. Passwords must contain at least one character (letter or digit) and no duplicate characters are allowed.

For example:
- `BIO14` is a valid password;
- `OLYMPIAD` is not a valid password (letters are not in alphabetical order).

The passwords have been put into an ordered list. If two passwords contain a different number of characters, the password with the fewer characters comes first. If they both have the same number of characters then they are sorted alphabetically.

The ordered list of passwords looks like this: `A`, `B`, ..., `Z`, `0`, `1`, ... `9`, `AB`, `AC`, ..., `A9`, `BC`, ...

3(a) [25 marks]

Write a program to determine the n^{th} password in the ordered list.

Your program should read in a single integer, ($1 \leq n \leq 1{,}000{,}000{,}000$). You should output the string which represents the n^{th} password.

Sample run

```
37
AB
```

3(b) [2 marks]

In what order do the following passwords appear in the list: `BIO`, `BIO14`, `NTU`, `14`, `ABCDE` ?

3(c) [3 marks]

How many passwords does the scheme accept?

3(d) [5 marks]

How many *pairs* of adjacent passwords (in the ordered list) contain all 36 characters between them, with each character appearing in only one of the passwords? Justify your answer.

Total Marks: 100 End of BIO 2014 Round One paper