British Informatics Olympiad

Time allowed: 3 hours

Sponsored by Jane Street

The 2022 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.
- Some written questions can be solved by hand without solving the programming parts.
- The final written part of each question is intended to be a difficult challenge.
- Do not forget to indicate the name given to your programs on your answer sheet(s).

Question 1: ***Decrypt***

Each letter in the alphabet can be given a value based on its position in the alphabet, A being 1 through to Z being 26. Two letters can produce a third letter by adding together their values, deducting 26 from the sum if it is greater than 26, and finding the letter whose value equals this result.

A simple *encryption* scheme for a string is as follows. Working from left to right, after leaving the left-most character alone, each character is replaced with the character produced by adding it to the character to its immediate left.

For example:

- The string ENCRYPT goes through the steps ENCRYPT → ESCRYPT → ESVRYPT → ESVNYPT → ESVNMPT -> ESVNMCT -> ESVNMCW

1(a) [24 marks]

Write a program that reads in an *encrypted* string, between 1 and 10 *uppercase* letters inclusive, and outputs the *original* version of that string.

Sample run

```
ESVNMCW
ENCRYPT
```

1(b) [2 marks]

Give a five letter string whose encrypted version matches the original.

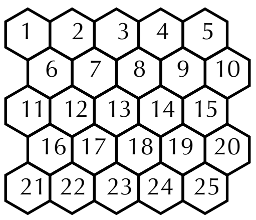
1(c) [2 marks]

How many times (≥1) must you encrypt OLYMPIAD before it becomes the original string again?

1(d) [4 marks]

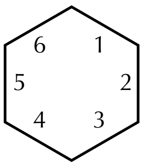
How many three letter strings, if encrypted 999,999,999,999 times, become the original string again?

Question 2: ***Game of Drones***



Two bee colonies are fighting for control of a *hive*.

The hive is represented by a 5x5 array of hexagons, numbered as in the diagram to the right. The six edges of each hexagon are numbered, as in the diagram below. Some edges are *owned* by either the *red* colony or the *blue* colony. If a colony owns more edges of a hexagon than another colony, that colony *controls* the hexagon. Initially, no edges are owned by either colony and hence no hexagons controlled.



Two drones are in the hive: a red drone on hexagon 1 facing edge 1 and a blue drone on hexagon 25 facing edge 6. Drones jump between hexagons in numeric order, returning to 1 after hexagon 25. Whilst jumping, drones do not change the direction they are facing.

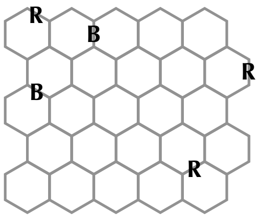
The game consists of a number of *skirmishes* followed by a number of *feuds*.

In each skirmish:

- The red drone takes ownership (for the red colony) of the edge it is facing, it then rotates 60° *clockwise* to face a new edge and finally it jumps *r* hexagons along the hive.
- The blue drone similarly takes ownership of the edge it is facing, it then rotates 60° *anti-clockwise* to face a new edge before finally jumping *b* hexagons.

For example, if *r* is 9 and *b* is 3:

- In the first skirmish, red takes ownership of edge 1 of hexagon 1 and blue takes ownership of edge 6 of hexagon 25.
- Red faces edge 2 (clockwise from 1) and jumps to hexagon 10 (1+9). Blue faces edge 5 (anti-clockwise from 6) and jumps to hexagon 3 (as 1 is the hexagon following 25).
- In the second skirmish, red takes ownership of edge 2 of hexagon 10 and blue takes ownership of edge 5 of hexagon 3.
- In the third skirmish, red takes ownership of edge 3 of hexagon 19 (which was previously owned by blue as that edge is shared with hexagon 25) and blue takes ownership of edge 4 of hexagon 6.

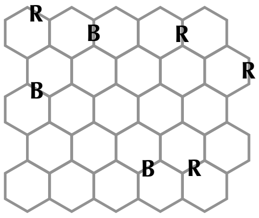


After the skirmishes it is time for feuding. In each feud:

- The red colony will take ownership of their preferred *un-owned* edge, followed by the blue colony doing likewise.
- When selecting an edge a colony *prefers* edges that gain them control over the most hexagons. Between edges that give them the same amount of control they prefer those that take away the most control from the other colony. After that, preference is based on hexagon number; lowest for the red colony and highest for the blue colony. After that, preference is based on direction number; lowest for the red colony and highest for the blue colony.

For example, continuing on from the *hive* after the previous skirmishes:

- Red takes ownership of edge 2 in hexagon 4. Taking ownership of this edge gives red control over hexagons 4 and 5. This is an edge in the lowest number hexagon that can gain red control over two additional hexagons. It is also the lowest edge in the hexagon that gains such control as edge 1 is only adjacent to a single hexagon. An edge in hexagon 1 cannot gain control of hexagon 1 (red *already* controls it) and at most removes the blue control from hexagon 2 or 6; a total gain of no hexagons. An edge in hexagons 2 or 3 would take control away from blue and, at most, gain control of either hexagon 4, 7 or 8; a total gain of control of only a single hexagon.
- Blue takes ownership of edge 6 in hexagon 24, similarly taking control of two hexagons.



2(a) [27 marks]

Write a program that plays the *Game of Drones*.

Your program should first input a line containing two integers, *r* ($1 \le r \le 25$) then *b* ($1 \le b \le 25$), indicating the number of hexagons the red and blue drones jump by during each of the skirmishes. This will be followed by a line containing two integers, *s* ($0 \le s \le 1000$) then *f* ($0 \le f \le 40$), indicating the number of skirmishes and feuds respectively.

Marks are available for cases where there are no feuds

You should output two lines, the first containing the number of hexagons controlled by the red colony and the second the number controlled by the blue colony.

Sample run

```
9 3
3 1
6
6
```

2(b) [3 marks]

Show the hive after 0 skirmishes and 7 feuds, making it clear which edges are owned by each colony.

2(c) [4 marks]

Suppose the drones continue skirmishing until an edge changes ownership between the two colonies. Assuming $1 \le r, b \le 25$ and the values are chosen so that an edge will eventually change ownership, what is the minimum number of skirmishes that take place before the skirmish where an edge changes ownership? What is the maximum number?

Question 3: ***Parking***

A one-way street, running left to right, has several parking spaces each large enough for a single vehicle. Each arriving car has a preferred parking spot and is driven along the street, ignoring empty spaces, until it reaches its preferred spot. If that spot is available it is parked there, otherwise it continues along the street and is parked in the first unoccupied space. If all those spaces are occupied it leaves the street and does not park anywhere.

Spaces are labelled `A`, `B`, … in the order they are encountered. Cars are labelled `a`, `b`, … in order of their arrival. There are the same number of cars as parking spaces and all the spaces are initially empty.

A *preference list* shows the preferred parking spaces for the cars (in order of their arrival).

For example, suppose after each car has arrived they are parked in the order `cabd`:

- Car `a`'s preferred space was `B`. When it arrived it drove past space `A` to space `B`, found it empty and parked.
- Car `b` might have a preferred space of `C` (driving past spaces `A` and `B` on arrival, finding space `C` empty and parking) but it might also have a preferred space of `B` (driving past `A` on arrival, finding `B` full and parking at the next available space `C`).
- Car `c`'s preferred space must be `A`.
- Car `d` might prefer any of the spaces. If their preferred space is `A`, `B` or `C` when they arrive they find it full and the first available space, in each case, will be `D`.
- There are eight difference preference lists for the cars: `BBAA`, `BBAB`, `BBAC`, `BBAD`, `BCAA`, `BCAB`, `BCAC` and `BCAD`. These have been listed in *alphabetical* order.

3(a) [25 marks]

Write a program to determine the i^{th} preference list for a given final arrangement of cars *where all the cars have managed to park*.

Your program should input a string of n ($1 \le n \le 16$) lowercase letters (a permutation of the first n letters of the alphabet) giving the final arrangement of the cars, followed by an integer i ($1 \le i \le 2^{63}$).

You should output the i^{th} preference list for the cars that will lead to the final arrangement.

Sample run

```
cabd 5
BCAA
```

3(b) [2 marks]

Suppose the preference list is `EECCGGAAAA`. What is the final arrangement of the cars?

3(c) [3 marks]

Consider the cases where there are 16 cars, which all park, and exactly two possible preference lists lead to the final arrangement. How many arrangements are there?

3(d) [4 marks]

The final arrangement for 16 cars is `abcdefghijklmnop` and exactly 3 cars are in their preferred position. How many potential preference lists are there?

Total Marks: 100 End of BIO 2022 Round One paper