Perse Coding Team Challenge

2020

Round 2 Trios

A maximum of two Year 10/11 students are allowed in any trio

There are 12 questions with 60 minutes allowed.

Three students will work together on three adjacent computers

This is meant to be fun – do make sure you take the odd pause to enjoy it! If you are in a younger year group then remember you can always use what you learn this year for another year to come.

SUBMISSION ESSENTIALS:

- Ensure you SET THE LANGUAGE of your coding editor on hackerrank once you start Q1 (top right of coding area) to the language you wish to use.
- Ensure your inputs contain NO PROMPTS; the only screen output should be as requested in the question.

INSTRUCTIONS:

- You should work in threes for Round 2 wherever possible (solo/duo if necessary).
- You will only be given ONE copy of this paper per team; printed single sided you can discuss/divide/conquer!
- Your teacher will invigilate you for precisely 60 minutes and cannot discuss the problems. If you are finding questions challenging then your teacher can silently point to the help guide, the question text or your screen on Q1-3 only.
- Each trio may use up to three computers for Round 2 and you may discuss problems/solutions quietly within your trio and so should be sitting next to each other, separated from other trios wherever possible.
- All code submissions must be made within the time allowed. Any team that submits code after their time has been ended by the invigilator will be automatically disqualified from the competition.
- You may write code directly into each hackerrank question page or you may write it in a development environment and copy/paste it across so long as these submissions are made within the competition time.
- You may use the formal language reference online documentation for your language and you may also bring up to 10 A4 pages (20 sides – physical or digital) of notes/snippets into the competition. You should not have access to any other applications or resources (in particular no calculators are allowed either physical or digital). Examples of possible notes/snippets are on our website.
- Questions may be attempted in any order, all questions are worth 10 points with each code submission tested against up to 10 test cases.
- You should have some rough paper and a pen handy.

NOTES:

- Ensure you are logged in to hackerrank on your machines before the time starts when you open the paper.
- The invigilating teacher will ask you for your team name and your hackerrank usernames (top right once logged in) for submitting your team details as soon as possible after the event. Your hackerrank usernames used should be anonymous: you can change your hackerrank username easily and should do so before the competition starts under settings → account settings. Inappropriate usernames will be disqualified.
- Full marks will only be awarded to those solutions which can complete within the processing time limit

LEVEL 1

1) SCREEN TIME

You like some down time with your screen, but you also know the recommendations and you put your phone away when doing homework and also stay away from it for **an hour** before going to bed. Each homework takes you **35** minutes. Dinner also takes **some time**, and of course you don't have your phone at the table.

You are given 4 inputs;

- The time (an hour) you get home from school in 24-hour notation e.g. 16 for 4pm.
- The time (an hour) you go to bed again in 24-hour notation e.g. 21 for 9pm.
- The number of homeworks you plan to do
- The number of minutes taken for dinner

You should output the maximum number of **minutes** of available screen time that you will have.

Example Input

16
20
2
30

Example Output

80

Example Explanation

You have 4 hours (i.e. 240 minutes) between 4pm and 8pm but 2 homeworks of 35 minutes each, along with 30 minutes for dinner and the pre-bed down-time takes it down to 80 minutes available.

Constraints

- The provided bedtime will always be before midnight

2) FIVE-A-DAY FILTER

You are regularly being told that you're not eating enough fruit and veg. You have a new method to decide when to stop eating the healthy food coming your way for that day. You start counting the food, with the first item being food number 1. As soon as the item of food has **fewer** letters than its number, then you don't eat it and you stop. You must output how many items of healthy food were eaten that day.

Example Input

Grape
Kiwi
Banana
Pea
Cabbage

Example Output

3

Example Explanation

Grape is item 1, and has 5 letters so that is eaten
Kiwi is item 2, and has 4 letters so that is eaten
Banana is item 3, and has 6 letters so that is eaten
Pea is item 4, but only has 3 letters so it is not eaten (3 < 4) and I stop accepting healthy food.
The number of items eaten was 3 (Grape, Kiwi, Banana)

Constraints

- It is guaranteed that you will **not** eat all the items provided in the input lines

3) BUT I'M THE OLDEST

The oldest always wins the argument. Input a list of 3 firstnames and ages (one firstname and age per line space separated) and output details of the oldest but in a tabulated form shown below.

Note that all text is left-aligned, the column headings NAME and AGE are in upper case and that the dash character (-) and pipe character (|) are used as row/column separators. The columns do not have fixed width but will be always be as small as possible.

Example Input

```
Jashina 19
Billie 28
Jack 27
```

Example Output

```
|NAME   |AGE|
|-------|---|
|Billie|28 |
```

Constraints

- All the ages provided will be distinct
- The provided firstnames will be between 2 and 50 characters in length inclusive
- The ages will be between 1 and 130 inclusive

LEVEL 2

4) ANAGRAM ADDITIONS

Input two lower-case words on two separate lines. The second word isn't an anagram of the first word because the first word hasn't got enough letters.

The rules of Anagram Additions are that you are allowed to duplicate any letter that you have as many times as you want. You can also request specific additional letters (which again you would then be allowed to duplicate).

Output the number of new letters that you would need to request in order to complete the anagram under these rules. Do not count any duplications that are needed.

Example Input

ship
physics

Example output

2

Example explanation

If the letters c, s and y were added to the letters in ship (s, h, i and p) then the word physics could be made. You already have an s however so you can duplicate this so you need only request 2 more letters. Note that if the required word had been psychotic then you would only have needed to request 4 because you could have duplicated the c once you had first requested it.

Constraints

- Each word will contain at least 1 and at most 100 letters

5) GROWING FLOWER

I have a special type of flower in my garden. It grows one unit higher for every day that it rains but it only flowers if it gets at least three **consecutive** days of some sunshine. Each day is coded as two adjacent letters: c or s followed by d or r:

c (cloudy all day) or s (some sunshine)
d (dry all day) or r (some rain)

Input the weather for the week as 7 lines of input and output a picture of my flower at the end of the week using:

```
***   for flowering – a maximum of one line of flowering if present
 |    for each unit of height
\_/   for my pot
```

For example the above drawing represents a plant of one unit high which has flowered. This would have occurred if it only rained once in the week and there were at least 3 consecutive sunny days. No more spacing is added to the output than is required. In particular the height is achieved by a single space and a pipe (|) but no further spaces.

A plant of two units high that hasn't flowered would be represented by:

```
 |
 |
\_/
```

Example Input

```
cd
sr
sr
sd
sd
sd
sd
```

Example Output

```
***
 |
 |
\_/
```

Example Explanation

Rain occurs on two days so it is two units high. Three consecutive days of sunshine do occur (six do in fact!) so the my plant has flowered.

Constraints

- There will always be at least one day of rain

6) RANDOM BANK

Most banks award interest at a percentage interest rate but Random Bank likes to do things differently. Instead for each $x you invest with them for one whole year, they will credit $y into your account on the end of the last day in the year.

For example imagine that x is 20 and y is 5.

If I invest $90 then I have four lots of $20 invested and the $10 will not be earning anything extra. So at the end of the year I will be credited with four lots of $5 namely $20 to my account and my balance for the start of the next year will be $110. If I invested for another year then I would then have five lots of $20 earning interest (and again no extra interest from the spare $10).

Input four positive whole numbers on separate lines. The first number will be the amount I invest, the second number will be the number of years I invest for. The last two numbers will be x and y as described above.

Output the amount of money that I have **made** at the end of the specified number of years, assuming I withdraw nothing in the meantime. The output should always be given without commas or spaces e.g. $10000 not $10,000.

Example Input

90
2
20
5

Example Output

$45

Example explanation

This example matches the example at the start of the question: in my second year I am credited with five lots of $5 and so my final balance is $135. I have therefore made $45 over the two years which is my output.

Constraints

- The amount of money on my account will never exceed $1,000,000,000
- 0 <= x <= y <= 1,000,000 where both x and y are integers
- The number of years is at least 1 and at most 1000

LEVEL 3

7) EXPLODING NUMBERS

With some specialist chemistry & physics expertise I have worked out how to explode numbers.
Each number, with hydrogen and a small spark applied, explodes in the following way:

- take each single digit in the number
- small digits (those under 5) don't react and retain their value
- larger digits (6+) react aggressively and are replaced with one more than the square of their number within the original number string
- these are chain reactions of course so any new digits of value 6+ will continue the explosion

Input one whole number (negative or positive) and output its final fully exploded form.

Example Input

53618

Example Output

5335013505

Example Explanation

Digits 5, 3 and 1 will not react but 6 will be replaced with 37 and 8 will be replaced by 65 to give an exploded form of 5337165.

However there are still digits of value 6+ and so it continues to give form 533501375 and then form 5335013505

This latest form only contains digits 0 - 5 however and so it is in its final exploded form.

Constraints

- inputs will be whole numbers n of maximum one million digits

8) BOUNCING BALL

In this simulation, a ball is moving on a diagonal path inside a rectangle, bouncing off the edges to continue a diagonal path and maintaining a constant speed. If the ball went into a corner, it would just come back out directly reversing its path. The grid spaces inside the rectangle will be represented with a dot . and the ball with the lower case letter o.

Input two space separated positive whole numbers m and n to specify the size of the grid m x n where m is the number of columns and n is the number of rows. In the below example, m = 5 and n = 4. Next input two space separated positive whole numbers x and y to specify the initial grid location of the ball measured from the top left location which has coordinate (0,0).

Next input two space separated whole numbers a and b to specify the initial speed vector of the ball. This can be one of the four diagonal possibilities: 1 1, 1 -1, -1 1, -1 1. For example, with a = 1 and b = -1, the initial speed means one square to the right and one square up for every 'step'. Finally input a positive whole number s to indicate the number of steps to simulate after the starting position. Output the representation of the grid (in the form shown below) after the s steps have been taken.

Example Input

```
5 4
1 2
1 -1
4
```

Example Output

```
.....
.....
...o.
.....
```

Example Explanation

Initial:

```
.....
.....
.o...
.....
```

After 1 step:

```
.....
..o..
.....
.....
```

After 2 steps:

```
...o.
.....
.....
.....
```

After 3 steps:

```
.....
....o
.....
.....
```

So after four steps, the output is as shown in the example output above.

Constraints

- 2 <= n, m <= 50
- 1 <= s <= 1×10^{13}

9) PASS IT ON

I have some good news so, after a minute's pause, I tell it to my contacts. They all pause for a minute and then tell any of their contacts who don't know already.

Assuming this process continues, output how long it will be (in minutes) until every person named anywhere in the input information knows the news?

Each person is represented by a different capital letter of the English alphabet. I am always the person 'I' and the first line of contact connections will always be mine. Every other person can appear in the input in any order and be represented by any other capital letter.

The first line of input will be a single positive number n representing the total number of people including me.

The next n lines of input will specify contact connections and will be of the following form:
X:ABCD...
meaning that X has person A, person B, person C and person D as their contacts (not necessarily in alphabetical order). If X had no contacts at all then this line would end with a colon.

Example Input

5
I:ABC
A:ID
B:C
C:IB
D:A

Example Output

2

Example Explanation

After 1 minute I tell my contacts A, B and C. After another minute, A tells D but B and C have no-one new to tell. So after 2 minutes everyone knows: the output is 2.

Constraints

- a maximum of 26 lines of contact connections will be entered (i.e. the largest n is 26)
- note that contact lists do not need to be reciprocal: note the above for example where B is on my (I's) contact list but I am not on B's
- It is guaranteed for provided inputs that the news will eventually reach all the people

LEVEL 4

10) LETTER CHOP

Capital letters of the alphabet are assigned numerical value in order from -12 for A, to +13 for Z.

Your input will consist of a string of capital characters in a single line.

Output the greatest total letter score which can be achieved from any consecutive substring. Output also, on the next line, the first and last characters from this maximal substring.

Example Input

GEUHRTNB

Example Output

16
UN

Example explanation

The substring UHRTN achieves the best score here. It is worth 8 + -5 + 5 + 7 + 1 = 16. This is the greatest achievable substring score and so the output is 16. The first and last characters of this substring are U and N and so UN is also output on a second line.

Constraints

- Input strings can be up to 200,000 characters in length and will contain only capital letters of the English alphabet
- At least one letter of positive value will always be used in every input string
- Each test provided will have a unique solution
- If the first and last character of the substring are the same letter then output it twice e.g. RR

11) ASTEROIDS

Bertie is working on creating a video game. The player controls a space station which shoots out laser beams to destroy incoming asteroids. The world is two-dimensional and each asteroid is modelled by non-overlapping polygons. The space station is located at the origin, coordinates (0, 0), and the player shoots by declaring target coordinates. The laser beam hits and destroys every asteroid that falls in its path (which is a ray originating from the space station and continuing indefinitely beyond its target). If the laser beam only hits a vertex or an edge of an asteroid, this asteroid STILL gets destroyed.

Your task is to work out which asteroids will eventually be destroyed by the laser beam.

The first line of the input contains two space separated non-negative integers: the coordinates of the target the player is aiming at. The second line of the input contains one positive integer: the number of asteroids.

The following lines describe the asteroids, one line per asteroid: the first number in the line specifies the number of vertices of this asteroid n_i followed on the same line by n_i pairs of numbers: the x and y coordinates of the asteroid's vertices. The vertices are always given in clockwise order. The first asteroid listed will have **index 1**, the second listed will have index 2 and so on.

OUTPUT:

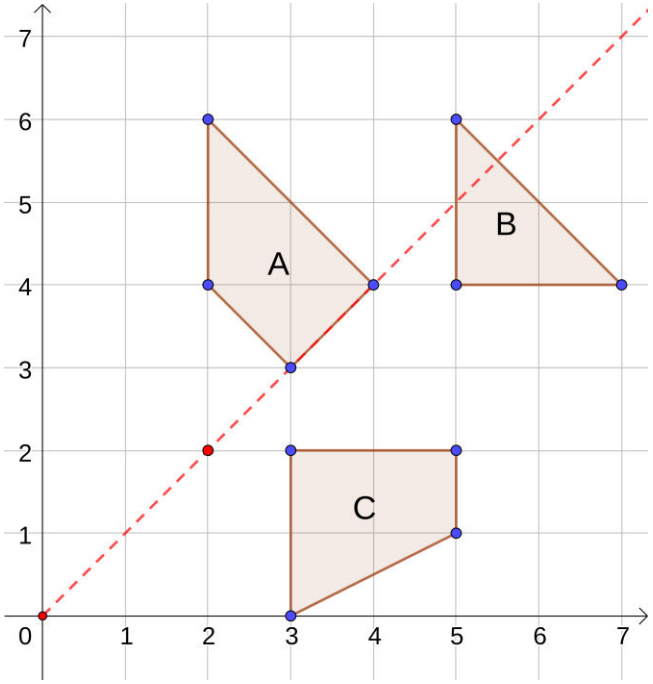
Output the index numbers of the asteroids hit by the laser beam, in ascending order, one per line.

Example Input

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

EXAMPLE OUTPUT

```
1
2
```



Explanation

See diagram above – asteroid A is the one listed first: it has index 2 and its definition line starts with 4 because it has 4 vertices. Asteroid B has index 2, asteroid C has index 3. A (index 1) and B (index 2) are hit by the ray and so the output is as shown. Note: the output indexes should be stated in ascending order regardless of the order in which the ray hits them.

Constraints

- No asteroid is covering the space station i.e. point (0, 0) does not belong to any asteroid
- Vertices provided will specify (x,y) such that 0 <= x, y <= 30,000
- The number of vertices of each asteroid n_i will satisfy 1 <= n_i <= 50
- Each asteroid's vertices will always be specified in clockwise order
- There will be at least one and at most 100 asteroids
- The laser beam will hit at least one asteroid

12) **NEARLY PERFECT SHUFFLE**

To nearly-perfectly shuffle a deck of *n* cards using *offset k*, perform the following steps:

- Remove *k* top cards from deck *D*, placing them on a new pile *P*, one at a time.
- Split the remaining *n*-*k* cards equally into the top half (*T*) and the bottom half (*B*).
- Place the bottom cards of *B* and *T* on top of *P* in this order.
- Repeat the previous step until all cards are on *P*, which becomes the shuffled deck.

For example:

n = 11, *k* = 3 with starting deck from the *bottom*: 3 7 9 A 2 8 J K 6 4 Q

First we place Q, then 4, then 6 on pile *P*.

Then we split the remaining cards into bottom half: 3 7 9 A and top half: 2 8 J K.

The bottom card of B is then 3 which we place on P followed by the bottom card of T which is 2

Repeating the previous step, we end up with the shuffled deck: Q 4 6 3 2 7 8 9 J A K.

Your task is to write a program which, for given values *n* and *k*, outputs how many nearly-perfect shuffles are needed to return a perfectly sorted deck back to its original state.

Example Input 1

6
0

Example Output 1

4

Example Input 2

11

3

Example Output 2

10

Constraints

- 3 <= *n* <= 1,000,000
- 0 <= *k* <= *n*
- *n* - *k* will always be even for provided inputs
- The required outputs will not exceed 10^{18}