The Perse Coding Team Challenge
2018

1 HOUR
[10 minutes team discussion + 50 minutes pairs programming]

THIS DOCUMENT MUST BE PRINTED SINGLE SIDED FOR QUESTION SEPARATION
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.

INSTRUCTIONS:

- You must be in teams of four or fewer with a maximum of 2 Year 10 students as the oldest.*
- Your teacher will invigilate you for precisely 1 hour and <u>cannot</u> discuss the problems with you.
- For the first ten minutes you will discuss the problems in your team. No notes must be made during this stage – verbal discussion only.
- Teams must divide the questions into two piles during the discussion stage.
- After ten minutes, you will split into two separated pairs taking one pile each. From then on no communication is allowed between the pairs.^
- You may pass <u>one and only one</u> unmarked question sheet either from pair A to pair B or vice-versa during the remaining 50 minutes via the invigilating teacher but no cross-pair discussion is allowed.
- You may use up to four computers in total and you may also discuss problems/solutions <u>within</u> each pair and so each member of a pair should be sitting next to each other.
- All code submissions must be made <u>within</u> the hour. Any team that submits code after their hour has been ended by the invigilator will be <u>disqualified</u> 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 hour. You may use the formal language reference online documentation for your language and you may also bring up to 10 A4 pages (20 sides) of notes/snippets into the competition so long as all team members have access to the same material. You should not have access to any other applications or resources (in particular no calculators are allowed either physical or digital).
- Questions may be attempted in any order. The marks available depend on the test cases passed. Higher level problems are worth more points with 10/20/30/40 points available for a Level 1/2/3/4 problem respectively.
- You should have some rough paper and a pen handy for the main 50 minute phase.

HINTS & NOTES:

- Ensure you are logged in to hackerrank before the hour starts on your machines to assist in a smooth transfer to the coding.
- 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.
- Note that any leaderboard scores shown on hackerrank are simply reflections of individual login scores. The ultimate contest scores are determined by your overall team result across your two pairs.

** For purposes of yeargroup classification, the student date of birth is used to classify them to their typical national yeargroup for that age.*
^ Teams of three must split into a two and a one. Teams of two may split into two singles or may stay as a single pair.

<u>Level 1 – Question 1 [10 points]</u>

THREE IS A HAPPY CROWD

Three whole numbers are said to be a HAPPY CROWD if any two of the numbers add up to the third. The numbers can come in any order. For example this set of inputs is a HAPPY CROWD:
4
11
7

INPUT:
3 lines of input, each line specifying a single whole number

OUTPUT:
Output either `HAPPY CROWD` (uppercase) if the three numbers form a HAPPY CROWD or `UNHAPPY CROWD` if they do not.

EXAMPLE INPUT:
```
-3
8
11
```

EXAMPLE OUTPUT:
```
HAPPY CROWD
```

INPUT FORMAT NOTES:
- all inputs are whole numbers between -1000 and 1000 inclusive

Level 1 – Question 2 [10 points]

ETHAN'S PRIMES

Ethan worked hard to develop a formula for prime numbers. After some time he realized that odd numbers are similar and easier to tackle than primes, so he decided to start with a slightly simpler problem. He would like to write a program to find the nth smallest positive odd number.

INPUT:
A positive whole number *n*

OUTPUT:
Output the *n*th smallest positive odd number

EXAMPLE INPUT:

```
8
```

EXAMPLE OUTPUT:

```
15
```

INPUT FORMAT NOTES:

- *n* will not exceed 10,000

Level 1 – Question 3 [10 points]

PERFECT CIPHER

Jemima tries to develop a cipher, which would be very difficult to decode. She finally has an idea! If she substitutes every letter of the message with ‘E’ (the most common letter in the English language), she is sure nobody will be able to decipher it!

INPUT:
A single word

OUTPUT:
Output the message encoded using Jemima’s cipher

EXAMPLE INPUT:
```
PERSECODING
```

EXAMPLE OUTPUT:
```
EEEEEEEEEEE
```

INPUT FORMAT NOTES:
- all characters are upper-case letters of the English alphabet
- the word will contain no more than 30 characters

Level 1 – Question 4 [10 points]

COUNTING VOTES

You were commissioned to write a program counting the votes in a school referendum. Your task is to count all the YES votes and all the NO votes and print the result. Ignore all spoiled votes (that is, votes that say anything else than YES or NO)

INPUT:
The first line of the input contains one integer *n* - the number of votes cast in the referendum.
Each of the following n lines contain one vote, written in uppercase letters of English alphabet.

OUTPUT:
In the first line output YES followed by the number of YES votes.
In the second line output NO followed by the number of NO votes.

EXAMPLE INPUT:
```
8
YES
NO
NO
NO
YES
ABSTAIN
NO
YEAH
```

EXAMPLE OUTPUT:
```
YES 2
NO 4
```

INPUT FORMAT NOTES:
- *n* is between 1 and 1000 inclusive.
- Each line of the input is at most 10 characters long.

<u>Level 2 – Question 1 [20 points]</u>

MARATHON

Percy took part in a marathon race recently. He downloaded the results, but they were in alphabetical order and he would like to know which position he finished in. Write a program to determine this.

INPUT:

The first line of the input contains a single positive whole number *n*.
Each of the following n lines contains a participant's name being a single word consisting of English letters followed by a space and this participant's time formatted as *h:mm:ss.d*

OUTPUT:

Output a single integer (between 1 and n inclusive) – Percy's finishing position.

EXAMPLE INPUT:

```
4
Albert 4:10:33.4
Brian 4:17:51.4
Percy 4:09:10.9
Ulrika 3:54:33.2
```

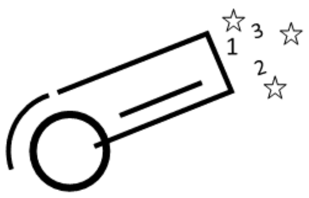
EXAMPLE OUTPUT:

```
2
```

INPUT FORMAT NOTES:

- assume that the names are at most 20 characters long and they are listed in alphabetical order.
- also assume there is exactly one Percy on the list and that his finishing time is unique.
- for *n* in input line 1, assume *n <= 10,000*
- for participants' times h:mm:ss.d assume *2<=h<=9, 00<=mm,ss<=59* and *0<=d<=9*

Level 2 – Question 2 [20 points]



THE NUMBER-CANNON

A number-cannon is having fun, shooting the positive whole numbers up into the top vertex of a very large triangle, all covered with glue to make them stick together.

The cannon shoots left-to right, shooting row after row, and so after the first six shots, the numbers look like this with three rows complete (only the top part of the triangle is shown).

```
   / \
  / 1 \
 / 2 3 \
/ 4 5 6 \
```

INPUT:
Read in a single positive whole number *n*

OUTPUT:
Output the total of all the stuck-up numbers once n ROWS of numbers are stuck up

EXAMPLE INPUT:
```
3
```

EXAMPLE OUTPUT:
```
21
```

INPUT FORMAT NOTES:

- assume *1 <= n <= 200*

EXAMPLE EXPLANATION:
After three rows, the numbers 1-6 are stuck up and 1+2+3+4+5+6 = 21

Level 2 – Question 3 [20 points]

THE GRACEHOPPER

A robot-cycle called Gracehopper is going to start from rest (zero-speed) and accelerate towards a ramp and attempt to jump over a number of buses. For each section of run-up track, Gracehopper will increase her speed by 3 m/s. She will keep her speed constant whilst on the ramp itself.

The stunt will be described by an input string such as +--/###\ which will consist of the following codes which will always appear in this order within the string:
1) Each plus (+) character is short for 5 sections of run-up track
2) Each hyphen (-) means a single section of run-up track
3) A single forward slash (/) means the ramp itself
4) Each hash (#) indicates a bus that follows the ramp
5) A single backward slash (\) will always end the input to indicate the landing ramp

Some input strings may not use both + and - codes (although there will always be some run-up track!). There will always be exactly one /, one or more # and exactly one \ to finish. The ramp is angled such that Gracehopper will be able to successfully jump one bus for every 4 m/s that she is travelling at when she hits the ramp.

INPUT:
Read in the stunt setup as a single line of text without spaces

OUTPUT:
Output either a single backward slash \ if the robot-cycle can make the landing or a string of # characters, whose length indicates the bus that she would crash into. ### for instance means that Gracehopper would crash into the third bus.

EXAMPLE INPUT 1:
```
+--/###\
```

EXAMPLE INPUT 2:
```
+--/######\
```

EXAMPLE OUTPUT 1:
```
\
```

EXAMPLE OUTPUT 2:
```
######
```

INPUT FORMAT NOTES:
- the input string will be a maximum length of one hundred characters

EXAMPLE EXPLANATION:
In example 1, there are 7 sections of run-up track (5+2). Gracehopper will hit the ramp at 21 m/s. That means she can successfully jump up to a maximum of 5 buses. There are three buses in her setup so she can land successfully on the landing ramp and so \ is output. If however the setup had included six buses, as in example 2, then the robot-cycle would have crashed on the sixth bus.

<u>Level 3 – Question 1 [30 points]</u>

PASS THE PARCEL

A game of pass the parcel is played where the parcel is passed to the holder's right once per second. Each time the music stops, the person with the parcel takes a layer of wrapping off. When they have unwrapped the parcel they hand it to the person on their left and they leave the circle. However, if they are the only player left, they can greedily tear off all the remaining paper and have the prize. The players' names are: Alice, Bob, Carl, Diana (etc.)

The players' names conveniently start with ascending letters of the alphabet! Alice always starts with the parcel, Bob is to her right and then Carl etc. The music is played for the same number of seconds each round.

INPUT:
You are given 3 numbers - *p,t,w* - on one line separated by spaces. assume that *1 <= p,w,t <= 13*
p is the number of players at the start of the game
t is the number of seconds that the music plays each time
w is the number of layers of wrapping on the parcel at the start of the game

OUTPUT:
You must print the first initial of the winner of pass-the-parcel - the person who removes the last layer of wrapping

EXAMPLE INPUT:	**EXAMPLE OUTPUT:**
`5 2 4`	`A`

EXAMPLE EXPLANATION:

```
Players are A B C D E
Parcel is   ^
Layers      4
```
(they are in a circle, so E is sitting next to A)

After 2 seconds the parcel is at C. C unwraps a layer, then passes it back to B and leaves the circle

```
Players are A B C D E
Parcel is       ^
Layers          3
```

After 2 more seconds the parcel is as follows (E unwraps a layer, passes it back to D and leaves)

```
Players are A B D E
Parcel is         ^
Layers            2
```

After 2 more seconds the parcel is as follows (B unwraps a layer, passes it back to A and leaves)

```
Players are A B D
Parcel is     ^
Layers        1
```

After 2 more seconds the parcel is as follows (D passes it to A, A unwraps it and wins)

```
Players are A D
Parcel is   ^
Layers      0
```

A is the winner (there are no more layers)

Level 3 – Question 2 [30 points]

ZERO-ONE NUMBERS

A zero-one number is a positive whole number with all its digits (in its decimal representations) being only 0s and/or 1s, for example 11, 1010 or 100000 (eleven, one thousand and ten, one hundred thousand)

Your task is to find the smallest zero-one number divisible by a given positive integer *n*.

INPUT:
A single positive whole number *n*

OUTPUT:
A single positive integer being the smallest zero-one number divisible by *n*

EXAMPLE INPUT:
```
34
```

EXAMPLE OUTPUT:
```
111010
```

INPUT FORMAT NOTES:

- you are guaranteed that such a number always exists
- for the input n, we have 2 <= n <= 100

Level 3 – Question 3 [30 points]

BOX OFFICE

The box office handles group bookings for a theatre. Each time a group calls in, it is allocated seats according to the following rules:

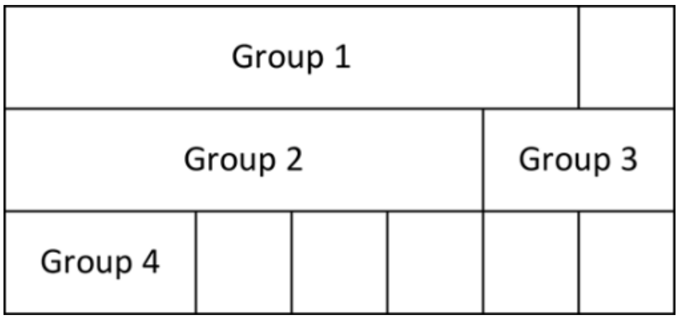
- The whole group must be seated in consecutive seats in a single row.
- If there is more than one row where the group could be seated, assign seats as close to the front as possible.
- If there are more free seats in the row, assign the leftmost seats.
- If it is impossible to seat the group according to the three rules above, the group will be turned away and sold no tickets.

INPUT:
The first line of the input contains two whole numbers *r* and *s* separated by a single space meaning, respectively, the number of rows in the theatre and the number of seats in each row.
The second line contains a single whole number *n* specifying the number of groups.

Each of the following n lines contains a single whole number g_i denoting the number of people in the *ith* group. Groups appear in the input in the order in which they come to the box office.

OUTPUT:
A single whole number being the total number of tickets sold

EXAMPLE INPUT:
```
3 7
5
6
5
2
2
6
```



EXAMPLE OUTPUT:
```
15
```

INPUT FORMAT NOTES:

- for input line 1, assume $1 <= r, s <= 200$
- for input line 2, assume $1 <= n <= 500$
- assume that each group size g_i satisfies $1 <= g_i <= 200$

Level 4 – Question 1 [40 points]

BEAT MY NEIGHBOUR

Here are some rules for the game Beat My Neighbour: Player X and Y start with an equal number of shuffled cards each (using a reduced pack for our simulation). Players will take turns, starting from player X, to place their top card face up onto a central pile. Play continues alternately until a 'payment card' (Ace, King, Queen or Jack) appears. When it does then the other player must 'pay' cards face up on to the pile using these payment rates: 4 cards for an Ace, 3 cards for a King, 2 cards for a Queen, 1 card for a Jack.

If the payment completes without another payment card appearing then the player who played the payment card collects the whole pile, turns the pile over face down and places it underneath their own pile. The player that wins the pile then goes first as the game continues with a new pile.

It often happens that while paying for a card, you turn over a pay card yourself. When this happens the previous pay card is cancelled and your opponent now has to pay for your new pay card. For example: X plays a queen; Y plays a 6 and then a Jack; so player X then has to pay and happens to lay an Ace. Y plays 3, 7, 4, 6. The Ace has been paid for in full so X takes the centre pile and places it on the bottom of their pile.

We will encode cards using single digits for each card value and using A, K, Q, J, T for Ace, King, Queen, Jack and Ten respectively. The suit will be ignored and there will be no Jokers.

INPUT:
Read in a first line to specify the initial hand of player X and player Y separated by a space given from top to bottom (i.e. top card on the left, bottom card on the right)
Read in a second line to specify a positive whole number n to represent the number of centre pile collections to simulate up to.

OUTPUT:
Output player X's hand after n centre pile collections. Cards should always be written with the top card leftmost.

EXAMPLE INPUT:
```
96K2J37 A334QJT
2
```

EXAMPLE OUTPUT:
```
79A6K334
```

INPUT FORMAT NOTES:

- assume that neither player will completely win before the requested number of collections has occurred (i.e. both players will always be able to play a card when required)
- assume that the initial 'hands' will be, together, up to a maximum of 52 characters in length, not including the single space between the two hands (i.e. a full pack)
- the number of centre pile collections requested will be between 1 and 100 inclusive

EXAMPLE EXPLANATION:
Player X starts with hand: `96K2J37` (with 9 on the top of their deck), Player Y has hand: `A334QJT`
X plays 9, Y plays A
X must pay up to four cards and so plays 6 and then K
Y must now pay and plays 3, 3, 4 and so X collects the pile (which face up is `433K6A9` with the 4 on the top so when turned face down is `9A6K334` with 9 on the top)

Player X now has hand: `2J379A6K334`, Player Y now has hand: `QJT`
X plays 2, Y plays Q
X plays J, Y plays J
X plays 3 and so Y collects the pile (which face up is `3JJQ2` and turned face down is `2QJJ3`)
Two pile collections have now been made and player X has hand `79A6K334` and so this is the output of the program (note that you only output player X's hand).

Level 4 – Question 2 [40 points]

BEACONS

The warning system of a faraway land consists of several beacons spread around the country. When a beacon warden sees another beacon alight, he lights his beacon (which takes exactly one minute).

The capital city, located at (0, 0) lights a fire to raise an alarm. Your task is to figure out how fast the alarm will spread to the rest of the country.

INPUT:

The first line of the input contains one positive whole number d, the largest distance in miles from which a beacon (or a fire in capital city) can be seen.

The second line of the input contains one positive whole number *n* the number of beacons, not including the starting beacon, which is always positioned at (0, 0).

Each of the following *n* lines of input contains two integers, separated by a single space, being the coordinates (in miles) of each respective beacon.

OUTPUT:

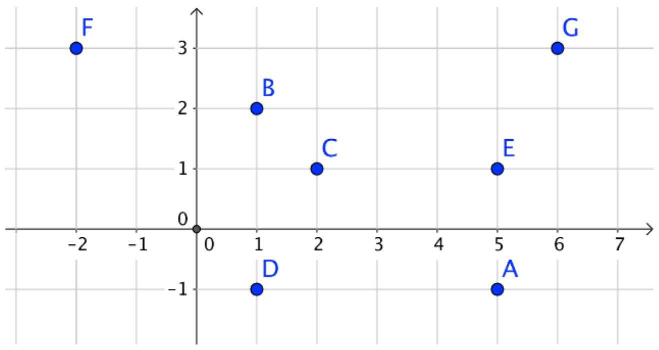
In the *mth* line of the output you should print the number of beacons alight after *m* minutes (not including the starting beacon). Your output should end when no more beacons would be lit in the following minute.

EXAMPLE INPUT:

```
3
7
5 -1
1 2
2 1
1 -1
5 1
-2 3
6 3
```

EXAMPLE OUTPUT:

```
3
4
6
```



INPUT FORMAT NOTES:

- assume that *1 <= d <= 10,000* where 10,000 is the largest distance in miles from which a beacon (or a fire in capital city) can be seen
- assume that *1 <= n <= 1,000*
- coordinate values are between -30,000 and 30,000

EXAMPLE EXPLANATION:

Beacons B, C, D would be lighted after one minute. Beacon E would follow in the second minute and beacons A and G after the third minute. Beacon F would never be lit.