TeamsCode Spring 2018 CGS Programming Contest

Problem Set

Rules:

1. Each question is worth 50 points. With each incorrect submission, the value of that question decreases by 5 points. You do not need to solve the problems in order.

2. The internet may not be accessed during the contest coding phase. Only one computer per team can be out during this time, meaning phones also need to be put away.

3. The teams with the highest score at the end of the contest will receive awards. In the case of a tie, the team that made their final successful submission first will be the winner.

Problems:

1. Hexagons
Input File: None.

Hexagons are the best polygons! Please output the pattern shown below (For convenience of reading, small spaces were inserted between underscores)

The hexagon overlords can see computer screens (egads!), so please ensure the pattern exactly matches the one shown below.

Input:

None.

Output:

You will output the pattern exactly as shown below.

Sample Output:

```
   ______
  / ____ \
 / / __ \ \
/ / /  \ \ \
\ \ \__/ / /
 \ \____/ /
  \______/
```

2. Anti-RPS

Input File: anti-rps.txt

As a prank, you decide to teach your friend the rules of *Rock, Paper, Scissors*, except that instead of the standard methods of deciding victory, you give them the reversed rules:

Paper beats Scissors
Rock beats Paper
Scissors beats Rock

Given a series of objects, please return the objects that beat them under the above ruleset.

Input:

A string consisting of `R` (rock), `P` (paper), or `S` (scissors).

Output:

A string consisting of the items that would beat the input string.

Sample Input:

```
RPSSPRRRRPS
```

Sample Output:

```
SRPPRSSSSRP
```

3. Tournament
Input File: tournament.txt

You are organizing a head to head meta-foosball competition. It has the format of a double round robin tournament, in which every entrant plays everyone else twice.

Since each match costs $5 to set up, large tournaments could get costly. Let the total cost of all the rounds be *C*. You will charge the lowest whole-dollar amount to each entrant such that the total amount collected from the entrants is at least half of *C*. Your sponsor will then pay the rest of the set up costs.

Input:

An integer, *n*, the number of lines.
For the next *n* lines:
An integer, a possible number of competitors.

Output:

The amount, in dollars, the sponsor will pay for each number of competitors; one amount per line.

Sample Input:

```
3
5
10
100
```

Sample Output:

```
50
220
24700
```

4. Artifacts

Input File: artifacts.txt

You are being chased by a horde of hangry raptors! You run through several rooms containing valuable artifacts you want to take. However, since there are raptors chasing you, you can only take one of the two artifacts per room. Each artifact has a value associated with it. You would like to maximize the total value of the artifacts you take.

Input:

An integer *c*, the total number of rooms.
A line with 2**c* space-separated integers, stating the values of the artifacts.

Output:

The largest possible total value of taken artifacts.

Sample Input:

```
4
5 10 17 18 16 16 30 1
```

Sample Output:

```
74
```

5. Cups and Swaps
Input File: cupsandswaps.txt

Have you ever seen the street performance in which a person places a coin under one of several cups, shuffles them, and invites an audience member to guess the location of the coin? Well, given the cup you want to track, you want to determine after a series of shuffles where the coin ends up.

There are *c* locations for the cups, numbered 1, 2, ... , *c*.

Input:
Three space separated integers, *c*, *t*, and *n*, the number of locations ($2 < c < 16$), the current location number of the cup to track (*t*), and the number of cup swaps ($0 < n \leq 50$).
For the next n lines:
Two space separated integers, *a* and *b*, specifying the locations of the two cups swapped.

Output:
The location of the tracked coin.

Sample Input:

```
8 1 10
1 2
3 2
3 4
5 4
5 6
7 6
1 2
2 1
8 6
7 8
```

Sample Output:

```
8
```

6. Chess

Input File: chess.txt

You want to play chess with a friend who doesn't know how the pieces move. You want to teach them how the queen moves by placing a white queen on a standard 8-by-8 square chessboard with a number of white pieces on it.

A queen can move to any square on a row, column, or diagonal it's on, as long as the move does not place it on the square of another piece and does not require it to jump over any piece.

Given a board of pieces, you would like to know to how many squares the queen can move.

Input:

An integer, *n*, which is the number of boards.
n boards, which are sets of 8 lines of 8 characters, "0" representing an empty square, "X" representing an occupied square, and "Q" representing the queen, with an empty line separating different boards.

Output:

For each board, print the number of squares the queen can move to.

Example Input:

```
1
00000000
00000000
00X00000
000QX000
00000000
00000000
000X0000
00000000
```

Example Output:

```
18
```

7. Outlier

Input File: outlier.txt

You have noticed that several problems on your previous tests took longer than expected. The instructions state that the time required for each problem should be close to a nondecreasing arithmetic sequence. What they don't state is how the fourth problem ended up taking more time than the last problem….

You talk to your classmates, who also have had this issue. They have called problems that take more time than they should *outliers*. You would like to identify these outliers.

Input:

Two space separated integers, n and x, specifying number of tests ($0 < n < 10$) and number of problems on each test ($3 < x < 10$).

For the next n lines, there are x space-separated integers, each integer specifying the time it took to complete the problem.

Output:

For each test, print the problem number (one more than the number of problems preceding it) of the outlier.

Sample Input:

```
4 5
1 2 15 4 5
2 2 2 3 2
1 4 3 4 5
10 9 10 11 12
```

Sample Output:

```
3
4
2
1
```

8. Street Crossing

Input File: streetcrossing.txt

You want to get to Costco to eat free samples of hummus, but there is a busy road to cross.

The road has several lanes, with cars intersecting your straight-line path every now and then.

Each lane takes two seconds to traverse, and you cannot stop in the middle of the street.
Assume that each car passes instantaneously.
You would like to determine the shortest amount of time to wait before crossing the road to avoid getting hit by a car.

Input:

An integer, *n*, the number of data sets.
For each data set:
An integer, *x,* stating the number of lanes.
Lanes in order of crossing: For the next x lines:
Two space-separated integers, *a* and *b*. Cars in the lane will pass every *a* ($2 < a < 60$) seconds, the first car is *b* ($0 \leq b < a$) seconds away from you.

Output:

The minimum time in seconds needed to wait to cross the road, and -1 if it is never possible to cross the road.

Sample Input:

```
1
4
5  2
7  2
9  2
11  2
```

Sample Output:

```
3
```

9. Trophies

Input File: trophies.txt

You have too many trophies from all the various competitions you won in the past, and you want to store trophies on your trophy stand into a box to create space on the stand. All your trophies are rectangular prisms with the same dimensions, with bases of 1 cm by *l* cm, and you would like to determine the number of ways to fit your trophies in a rectangular box with dimensions *l* cm by *s* cm.

Input:

An integer, *n*, the number of lines.
For the next *n* lines:
Two space-separated integers, *l*, and *s*, the length ($2 < l < 10$) of each trophy (the width and height are 1 cm), and box side length ($1 < s < 100$).

Output:

For each line, print the number of possible ways to arrange the trophies in the box. Two arrangements are different if the final layout of the trophies are different; the order the trophies are placed in the box does not differentiate between arrangements.

Example Input:

```
1
2  4
```

Example Output:

```
5
```

10. Password

Input File: password.txt

Congratulations! You've forgotten your password for the 71st time!

You only remember that your password consists of a series of strings and was a certain length... oh well. Time to make the next password "bubbleSortIsTheBest1234567890"...

You would like to know how many passwords exist such that each password consists of only strings from a given *allowed* list and is a specified length.

Input:

An integer, *n*, the number of data sets:
For each data set:
Two space-separated integers, *a* and *l*, specifying the length of the allowed list ($0 < a < 6$) and the length of the password ($0 < l < 10$).
a space-separated strings, each a string in the allowed list.

Output:

The number of possible distinct passwords.

Example Input:

```
2
3 9
a pass xyz
4 4
a b c d
```

Example Output:

```
40
256
```

11. Sequence

Input File: sequence.txt

As part of your job as a worker at a transmissions company, you receive several sequences. Your task is to determine if a sequence received is made of alternating prime numbers and perfect squares. Each valid sequence can start with either a prime or a perfect square, and has at least one of each.

Input:

An integer *x* stating the number of sequences
For each of the next x lines:
A sequence.

Output:

For each sequence, output "True" if the sequence consists of alternating prime numbers and perfect squares and has at least one of each, and "False" if not.

Example Input:

```
3
2424225
64322529
100816449
```

Example Output:

```
True
True
False
```

12. Tower

Input File: tower.txt

You are attempting to build a tower with several three-dimensional blocks in your collection of rectangular prisms. You want to know the tallest tower you can build, given that each brick must be equal or less in both volume and height than the brick immediately below it. You can choose to not use bricks if desired, and bricks may be oriented in any fashion.

Input:

An integer *x* stating the number of blocks.
For the next *x* lines:
Three space-separated integers, stating the dimensions of the block.

Output:

The height of the tallest constructable tower following the above rules.

Example Input:

```
6
1 1 1
2 2 2
4 10 4
8 6 3
5 6 7
20 20 20
```

Example Output:

```
41
```

13. Traffic Lights

Input File: trafficlights.txt

You are trying to drive across downtown Landport, from the northwest corner to the southeast corner. Of course, you have terrible traffic light luck, as when you started driving, all the traffic lights turned red. At each intersection, there is a traffic light that alternates between red for a number of minutes between 0 and 9, and green for 1 minute. If a light is red, you cannot pass through it. If a light is green, you can pass through it. Note that the direction a vehicle enters an intersection does not affect its passability. If moving between intersections takes 0.9 minutes, what is the shortest amount of time it would take to reach your destination?

Input:

An integer, *n*, the number of data sets.
For each data set:
An integer *x* ($0 \leq x \leq 9$), the size of the *x* by *x* grid. There are x^2 intersections.
For the next *x* lines:
x numbers indicating the length of time each red light takes.

Output:

For each data set, the fewest number of minutes needed to reach the destination, rounded to the nearest tenth.

Example Input:

```
1
4
5290
9681
0751
1080
```

Example Output:

```
10.8
```

14. Mirrors and Lasers
Input file: mirrorslasers.txt

You arrive at the corner (0, 0) of a two dimensional room, holding a laser pointer. There are mirrors covering the walls, and the sign outside the room says to aim the laser pointer to attempt to hit the sensor in the opposite corner of the room (l, w) in the fewest reflections possible. There are several double sided mirrors in the room, each with non reflective edges (beams are not affected by edges of mirrors). You are thinking of several angles at which to aim the laser pointer and would like to know how many reflections each path would take.

Input:

Four space separated integers, *l, w, m,* and *n,* specifying the length ($1 < l < 100$) and width ($1 < w < 100$) of the room, number of mirrors ($0 < m < 15$), and number of angles to test ($0 < n < 10$).
For the next n lines:
Four space-separated integers, *x1, y1, x2, y2,* specifying the location of the endpoints [(*x1, y1*), (*x2, y2*)] of the double sided mirror.
For the next *m* lines:
Two space separated integers, x and y, specifying the coordinate the laser is initially pointed towards.
All coordinates in the input are guaranteed to be non negative integers.

Output:

For each angle, print the number of reflections the laser will take to reach the sensor, or -1 if the laser does not hit the sensor.

Example Input:

```
10 10 1 3
0 5 4 5
1 1
1 2
8 10
```

Example Output:

```
0
2
-1
```

15. Battery
Input File: battery.txt

Someone forgot to charge the laptop batteries overnight, and, even worse, most of the chargers are missing! You would like to charge the many laptops as quickly as possible in preparation for the coming day. Each remaining charger delivers energy at a different rate, and each laptop has a different percent battery. Once a laptop is plugged in, it cannot be unplugged until fully charged. Assume an instantaneous switch between laptops after a finished charge, and that a laptop can only be charged with one charger at a time.

Input:

An integer, *n*, the number of data sets.
For each data set:
Two space-separated integers, *c* and *l*, specifying number of chargers ($0 < c < 3$) and number of laptops ($0 < l < 500$).
c space-separated integers, each integer signifying the number of minutes it takes for the charger to charge a laptop battery one percent.
l space-separated integers, each integer (between 0 and 100 inclusive) signifying the current battery percentage of the laptop.

Output:

The shortest amount of time, in minutes, to fully charge all laptops.

Example Input:

```
1
2 6
2 3
92 94 93 95 91 89
```

Example Output:

```
56
```