r/6thForm

> Programming Problems

This document contains a collection of programming tasks, with varying levels of difficulty.
This is *not* a programming tutorial. Instead, use it as practice to test your understanding.

How difficult are they?

Each of the problems have been given a difficulty rating, from ☆ to ☆☆☆☆☆.
In reality, all of the scenarios have an **accessible first task** - the rating only reflects the hardest in the set.

I'm not sure how to tackle one of the tasks.

Don't worry if the solution doesn't come immediately!

For tasks which involve large numbers or a variable amount of data, I'd recommend considering the **smallest test cases** you can. For tasks which require a lot of coding or need advanced data structures, I'd suggest jotting down a **rough outline** of your plan, potentially even **tracing out some of the steps** involved.

Where can I discuss the problems?

Whether you're one step away, need a hint, or hopelessly stuck, feel free to chat in `#computer-science`!
There are a **number of different approaches** you can take, so don't feel like there's one specific answer.

How often will this be updated? Will there be more problem sets?

I'll make sure to update the wording of and make corrections to existing sets. New problems will be added based on demand - it took quite a bit of work to get even the first five set up on my own, so I'd really appreciate any **feedback, suggestions or even ideas for new tasks** sent my way.

Good Luck!
- Keegan

★ | Swapper

- Let `words` be an array of strings.
- Let a be a non-negative integer, smaller than the size of `words`.
- Let b be a non-negative integer, smaller than the size of `words` and distinct from a.

Create a subroutine, `swap`, which takes `words` and swaps the item at position a with the item at position b.

For example, if our inputs are `a = 0, b = 2,` then the items in `words` get swapped like this:

```
["never", "gonna", "give", "you"] → ["give", "gonna", "never", "you"]
```

1. Create a working implementation of the subroutine.

Don't overcomplicate it - you only need 4 lines of code.

By applying `swap` repeatedly, create a subroutine which...

2. Shifts all elements in the array to the right by one, moving the last element to position 0.

```
["never", "gonna", "give", "you"] → ["you", "never", "gonna", "give"]
```

3. Reverses the order of elements in the array.

```
["never", "gonna", "give", "you"] → ["you", "give", "gonna", "never"]
```

4. Orders the array alphabetically.

```
["never", "gonna", "give", "you"] → ["give", "gonna", "never", "you"]
```

★★ | Filter

Write a script that takes in a list of numbers as input, and returns...

1. A filtered copy with no multiples of 5.

```
[0, 3, 5, 8, 10, 11, 14, 19, 20] → [3, 8, 12, 14, 19]
```

2. A filtered copy without any prime numbers.

```
[0, 3, 5, 8, 10, 11, 14, 19, 20] → [0, 8, 10, 14, 20]
```

3. A filtered copy excluding all elements that are the sum of two others in the list.

```
[0, 3, 5, 8, 10, 11, 14, 19, 20] → [0, 3, 5, 10, 20]
```

★★★ | Whitespace

You are given a string which has been corrupted, and stripped of all whitespace and punctuation.
Using a valid dictionary of words, you want to reconstruct the correct spacing.

Assuming our dictionary is `[“eye”, “eyes”, “tick”, “stick”]`, we can reconstruct:

`“eyestick” → “eye stick” or “eyes tick”`

Let the dictionary be:

```
[a, ail, an, and, ash, derail, era, gran, grand, grandma,
     mash, ran, rail, red, redder, shred, shredder]
```

1. By observation (don’t write a script), find all valid reconstructions of

   `“grandmashredderail”`

Here is one attempt at an algorithm to determine whether a corrupted string can be reconstructed:

A. Begin at the first character in the corrupted string.

*B. Find the **longest** prefix which forms a valid word in the dictionary.*
*If none exist, return **False**.*

C. Remove the prefix from the string, and return to Step B.
*If the string is empty, return **True**.*

2. Give an example of a dictionary and corrupted string for which this algorithm fails.
   Replace longest with shortest. Show that this new algorithm still fails.

---

3. Write a script that, given a dictionary and corrupted string, prints all possible reconstructions.
   Using an appropriate data structure, ensure that checking if a substring is valid takes *constant time*.

★★★★ | Collatz

Consider the following algorithm:

A. *Choose a positive, whole number.*

B. *If it's even, divide by two.*
If it's odd, multiply by three and add one.

C. *Take the resulting number, and repeat Step B.*

In 1937, Lothar Collatz claimed that no matter the starting number, the result will eventually reach 1.

$$3 \rightarrow 10 \rightarrow 5 \rightarrow 16 \rightarrow 8 \rightarrow 4 \rightarrow 2 \rightarrow 1$$

So far, no-one has been able to prove or disprove this.

1. By creating a script that terminates, show that the statement holds for the first 5000 numbers.

We say that the strength of a number is how many times Step B occurs before reaching 1.
For example, the strength of 3 is 7, the strength of 170 is 10, and the strength of 1 is 0.

2. Which of the first 5000 numbers has the highest strength? What is its strength value?
   If there is a tie, choose the lowest.

Even though we've limited ourselves to the first 5000 starting numbers, we'll still encounter larger ones (sometimes much larger) in our sequence. For example:

$$1999 \rightarrow 5998 \rightarrow 2999 \rightarrow 8998 \rightarrow 4499 \rightarrow 13498 \rightarrow \ldots$$

3. Determine the highest number encountered during the algorithm.
   What percentage of *unique* numbers encountered are above 5000?
   Leave your answer to 2 decimal places.

---

4. Modify your script to instead consider the first 1,000,000 starting numbers.
   Give answers to Task 2 and Task 3 with these updated conditions.

You probably noticed your script running significantly slower, and may have had to make a few optimisations.
(On my machine, this took around 25 seconds)

5. Using data structures, re-write your script such that no number is encountered more than twice.

★★★★★ | Packing

Welcome to Sahara, the world's latest and greatest online retailer. We're happy to work with you, and hope you'll enjoy your time here at the logistics department!

There's a bunch of square-shaped glass tables in the warehouse that we haven't got around to shipping. All are the same size, and they need to be placed next to each other in Loading Area A, which also happens to be a square. We're not asking you to lift them, but we need to know how many we can fit there at once. Glass tables are fragile, so we can't stack them up or flip them on the side - but, we can still find the most efficient way to arrange them.

Given that each table has side length `t`, and Loading Area A has side length `l`,

1. Write a function that returns the maximum number of tables which can fit into the loading area.

Alright, thanks! We've had a large delivery request for some ceramic plate sets. These are also pretty fragile, so don't think about doing anything that might break em'. We've packed them up in rectangular boxes of the same size, and need to know how many we can fit in Loading Area C, which is also rectangular.

Given that each box has dimensions `box_x` by `box_y`, and Loading Area C is `load_x` by `load_y`,

2. Write a function that returns the maximum number of boxes which can fit into the loading area. Keep in mind that the boxes can be rotated.

Nice work! Alright, last task - we've got some unspecified, cuboid-shaped packages, again all the same size and ready to be carried onto the trucks in Loading Area E. Unfortunately we don't know how many trucks we'll need, but good news - the packages aren't fragile, so feel free to arrange them how you want!

Given that each package has dimensions `pack_x` by `pack_y` by `pack_z`,
and the storage space in each truck has dimensions `truck_x` by `truck_y` by `truck_z`,

3. Write a function to determine the minimum number of trucks needed to transport all 500 packages. Keep in mind that packages can be stacked, flipped or rotated in any direction. Good Luck!