Spring 24 Div III Week 2 - Solutions

(taken from editorials of original problems)

Problem A

(Source: Original Problem)

Implementation problem! Read n from the standard input, and then n strings, convert each one to upper case (all languages have something that does this) and then print.

Problem B

(Source: Codeforces Round 898 Div. 4 Problem B)

Just brute force all possibilties for the digit to increase, and check the product each time. The complexity is $\mathcal{O}(n^2)$ per testcase.

You can make it faster if you notice that it's always optimal to increase the smallest digit (why?), but it wasn't necessary to pass.

Problem C

(Source: Codeforces Round 898 Div. 4 Problem C)

You can just hardcode the values in the array below, and iterate through the grid; if it is an `X`, we add the value to our total. See the implementation for more details.

$$\begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 1 \\ 1 & 2 & 3 & 3 & 3 & 3 & 3 & 3 & 2 & 1 \\ 1 & 2 & 3 & 4 & 4 & 4 & 4 & 3 & 2 & 1 \\ 1 & 2 & 3 & 4 & 5 & 5 & 4 & 3 & 2 & 1 \\ 1 & 2 & 3 & 4 & 5 & 5 & 4 & 3 & 2 & 1 \\ 1 & 2 & 3 & 4 & 4 & 4 & 4 & 3 & 2 & 1 \\ 1 & 2 & 3 & 3 & 3 & 3 & 3 & 3 & 2 & 1 \\ 1 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

The time complexity — $\mathcal{O}(1)$ per testcase.

Problem D

(Source: Codeforces Round 898 Div. 4 Problem D)

The key idea is greedy. Let's go from the left to the right, and if the current cell is black, we should use the operation starting at this cell (it may go off the strip, but that's okay, we can always shift it leftwards to contain all the cells we need it to).

We can implement this in $\mathcal{O}(n)$: iterate from left to right with a variable i, and when you see a black cell, you should skip the next $k-1$ cells (because the eraser will take care of them) and increase the number of operations by 1. The answer is the total number of operations.

Why does it work? Notice the order of operations doesn't matter. Consider the leftmost black cell we erase. It means none of the cells to its right are black. So it doesn't make sense to use the operation on any of the cells to its right, since they are already white. It is at least as good to use the operation starting at this cell and to the $k-1$ cells on the left, since we may or may not hit another black cell.

Problem E

(Source: Original Problem)

Sort the integers using an efficient sorting algorithm (the built in sort from your favorite language is perfect). Now all the repeated elements are in consecutive positions. Iterate through the elements one by one and append each to a new list/array if it differs from the previous element.

Problem F

(Source: Northwestern European Regional Programming Contest 2020 Problem C)

Problem

For n numbers between 0 and 100 you are given the average of all numbers (d), and the average of a subset of k of those numbers (s). Compute the average of the remaining numbers.

Solution

- The sum of all numbers is $d \cdot n$.
- So the sum of the remaining numbers is $d \cdot n - s \cdot k$.
- That parts contains $n - k$ numbers, so the average of those numbers is $(d \cdot n - s \cdot k)/(n - k)$.
- When the average is < 0 or > 100, print impossible.

Gotchas

- Precision issues, e.g. answers just below 0 or just above 100

Problem G

(Source: Nordic Collegiate Programming Contest 2013 Problem A)

Problem

Find an optimal planting schedule to minimize the earliest date when all the trees are mature.

Insight and Solution

- **Insight:** It is optimal to plant the trees in descending order of growth times.
  - **Proof**: If any two trees are not in this order, then swapping them cannot increase the result.
- **Solution:** Sort the trees by decreasing growth times, and output $\max(\text{position} + \text{growth time} + 1)$.
- **Time complexity:** $O(n \log n)$.