Coding w/ Vihaan

A coding and computers channel.

USACO Jan 2025

Bronze

Problem 1 - Astral Superposition

- Bessie is using her nifty telescope to take photos of all the stars in the night sky. Her telescope can capture an $N \times N$ $(1 \leq N \leq 1000)$ photo of the stars where each pixel is either a star or empty sky. Each star will be represented by exactly one pixel, and no two distinct stars share the same pixel.

- Overnight, something strange happens to the stars in the sky. Every star either disappears or moves A pixels to the right, and B pixels downwards $(0 \leq A, B \leq N)$. If a star disappears or moves beyond the photo boundary, it no longer appears in the second photo.

- Bessie took photos before and after the shifting positions, but after experimenting in Mootoshop, she accidentally superimposed one photo onto the other. Now, she can see white pixels where both photos were empty, gray pixels where stars existed in exactly one photo, and black pixels where there was a star in both photos. Bessie also remembers that no new stars moved into the frame of the second photo, so her first photo contains all of the stars in the night sky.

- Given the final photo, determine the minimum possible number of stars in the sky before the shifting incident for T $(1 \leq T \leq 1000)$ independent test cases. If no arrangement of stars can produce the given final photo, output -1.

Problem 1 - Astral Superposition

- **INPUT FORMAT (input arrives from the terminal / stdin):**
  - The first line of input contains T and T test cases will follow.
  - The first line of each test case will contain $N\ A\ B$.
  - Then follow N lines each representing one row of the superimposed photo. The ith row from the top is represented by a string $c_{i,1}\ c_{i,2} \cdots c_{i,N}$, where each $c_{i,j} \in W, G, B$, representing the colors white, gray, and black, respectively.
  - It is guaranteed that the sum of N^2 over all test cases will not exceed 10^7.
- **OUTPUT FORMAT (print output to the terminal / stdout):**
  - For each test case, output the minimum number of stars that existed before the shifting, or -1 if impossible.
- **More details: https://usaco.org/index.php?page=viewproblem2&cpid=1467**

Astral Superposition - Solution

- **Subtask 1**
  - When $A = B = 0$, the answer is simply the number of non-white pixels. Black pixels correspond to the same star appearing in both photos, and gray pixels correspond to a star appearing in the first photo but disappearing from the second.
- **Subtask 2**
  - Since stars only move within each row, we can solve each row independently. For each row, we can try all 2^N possibilities to assign stars to each position within the row, and check whether it is consistent with the output. One way to generate all such possibilities is by using bitmasks. For each possibility, the following conditions must be checked.
    - A black pixel implies that there must be stars at that location as well as the location to the left of it.
    - A white pixel implies there must not be a star at that location.
    - A gray pixel implies that there must be a star at that location or the location to the left of it.
  - The final step is to output the sum of the minimum answers across rows, or -1 if any row is impossible. The runtime per test is $O(N^2 2^N)$.

Astral Superposition - Solution

- **Subtask 3**
  - Note that black and white pixels immediately imply that certain locations do or do not contain stars, so let's deal with them first and the gray pixels later. More specifically,
    - Place all stars required to satisfy black pixels.
    - Check whether there are any unsatisfied white pixels.
    - Add stars to satisfy gray pixels, without causing any white pixels to be unsatisfied.
  - The first two steps are straightforward, while the last can be accomplished using a greedy approach. While there is an unsatisfied gray pixel, consider the leftmost such gray pixel - then we must choose to place a star in at least one of two locations (at the gray pixel, or to the left of it). It is always optimal to place a star at the gray pixel due to the following reasons:
    - The location to the left of that pixel might be outside of the grid or correspond to the white pixel, so it might not be available.
    - A star at that gray pixel can potentially satisfy a gray pixel to the right, whereas a star to the left of that gray pixel will not.
  - This approach can be implemented with two left-to-right passes over each row of the input grid, the first for the black pixels and the second for the remaining pixels. The runtime per test is $O(N^2)$.

Astral Superposition - Implementation

- Can be found at https://github.com/VihaanAnand/Coding-Vihaan/blob/main/usaco-2025-01/bronze/1.cpp

Problem 2 - It's Mooin' Time II

- Farmer John is trying to describe his favorite USACO contest to Elsie, but she is having trouble understanding why he likes it so much. He says "My favorite part of the contest was when Bessie said 'It's Mooin' Time' and mooed all over the contest."

- Elsie still doesn't understand, so Farmer John downloads the contest as a text file and tries to explain what he means. The contest is defined as an array of N ($1 \leq N \leq 10^6$) integers $a_1, a_2, \cdots, a_N$ ($1 \leq a_i \leq N$). Farmer John defines a moo as an array of three integers where the second integer equals the third but not the first. A moo is said to occur in the contest if it is possible to remove integers from the array until only the moo remains.

- As Bessie allegedly "mooed all over the contest", help Elsie count the number of distinct moos that occur in the contest! Two moos are distinct if they do not consist of the same integers in the same order.

Problem 2 - It's Mooin' Time II

- **INPUT FORMAT (input arrives from the terminal / stdin):**
  - The first line contains N.
  - The second line contains N space-separated integers $a_1, a_2, \cdots, a_N$.
- **OUTPUT FORMAT (print output to the terminal / stdout):**
  - Output the number of distinct moos that occur in the contest.
- **More details: https://usaco.org/index.php?page=viewproblem2&cpid=1468**

It's Mooin' Time II - Solution

- **Subtask 1**
  - We can loop through every pair of numbers from 1 to N and perform a linear scan to check whether a moo with those numbers appears in the contest. This takes $O(N^3)$ time, which is fast enough for $N \leq 10^2$.
- **Subtask 2**
  - Imagine that some integer o appears more than two times. Any moo of the form $[m, o, o]$ that exists can always be constructed using the rightmost two occurrences of o in the original array.
  - This motivates the following $O(N^2)$ solution - scan over the array from right to left and keep track of how many times each integer has been seen. When we see an integer for the second time, count the number of distinct different integers to the left of that one. The sum of these counts is therefore the answer.

It's Mooin' Time II - Implementation

- Can be found at https://github.com/VihaanAnand/Coding-Vihaan/blob/main/usaco-2025-01/bronze/2.cpp

Problem 3 - Cow Checkups

- Farmer John's N ($1 \leq N \leq 7500$) cows are standing in a line, with cow 1 at the front of the line and cow N at the back of the line. FJ's cows also come in many different species. He denotes each species with an integer from 1 to N. The i'th cow from the front of the line is of species a_i ($1 \leq a_i \leq N$).
- FJ is taking his cows to a checkup at a local bovine hospital. However, the bovine veterinarian is very picky and wants to perform a checkup on the i'th cow in the line, only if it is of species b_i ($1 \leq b_i \leq N$).
- FJ is lazy and does not want to completely reorder his cows. He will perform the following operation exactly once.
    - Select two integers l and r such that $1 \leq l \leq r \leq N$. Reverse the order of the cows that are between the l-th cow and the r-th cow in the line, inclusive.
- FJ wants to measure how effective this approach is. For each $c = 0 \cdots N$, help FJ find the number of distinct operations (l, r) that result in exactly c cows being checked. Two operations (l_1, r_1) and (l_2, r_2) are different if $l_1 \neq l_2$ or $r_1 \neq r_2$.

Problem 3 - Cow Checkups

- **INPUT FORMAT (input arrives from the terminal / stdin):**
  - The first line contains an integer N.
  - The second line contains $a_1, a_2, \cdots, a_N$.
  - The third line contains $b_1, b_2, \cdots, b_N$.
- **OUTPUT FORMAT (print output to the terminal / stdout):**
  - Output $N+1$ lines with the i-th line containing the number of distinct operations (l, r) that result in $i-1$ cows being checked.
- **More details: https://usaco.org/index.php?page=viewproblem2&cpid=1469**

Cow Checkups - Solution

- **Subtask 1**
  - It suffices to loop through the endpoints of every subarray, manually reverse it, then count the number of indices such that $a_i = b_i$. This runs in $O(N^3)$ time.
- **Subtask 2**
  - First, let's calculate the number of indices i where a_i is already equal to b_i. Let's denote this number as S. Note that if we are not reversing a subarray that contains i, this index will always contribute to the number of total checkups.
  - We can visualize reversing an array as a series of swaps. Suppose we want to reverse an array b of length M, then it is equivalent to swapping b_i with b_{M-i+1} over all $1 \leq i \leq \lfloor \frac{M}{2} \rfloor$.
  - Suppose we reverse an odd length subarray of length M. Let K represent the index of the middle element of the subarray. We are essentially swapping the $(K-1)$'th element with the $(K+1)$'th element, $(K-2)$'th element with the $(K+2)$'th element, $\cdots$, $(K - \frac{M-1}{2})$'th element with the $(K - \frac{M+1}{2})$'th element.
  - This motivates a solution where we enumerate over K, enumerate over i where we swap the $(K-i)$'th element with the $(K+i)$'th element of a, then determine whether $a_{K-i} = b_{K-i}$ or $a_{K+i} = b_{K+i}$. However, it may be the case that the elements that we swapped are already equal to their corresponding elements in b, and after the swap, they are no longer equal. Let x denote the number of integers j among $\{K-i, K+i\}$ that such that $a_j = b_j$ after the swap, and y denote the number of integers j among the set such that $a_j = b_j$ before the swap. We must add x to S to account for the new cows that can be checked up, and subtract y from S to account for the old cows that cannot be checked up. This way, we are always able to keep track of the total number of cows that can be checked up when enumerating.
  - Even-length subarrays can be enumerated similarly. The only difference is that the middle element is technically between the $\frac{M}{2}$'th and the $(\frac{M}{2}+1)$'th element. Let K represent the index to the left of the middle of the subarray, then we are swapping the $(K-i+1)$'th element with the $(K+i)$'th element for each i. We can track the total number of checked cows in the same way as odd-length subarrays.
  - Enumerating over all K and i over all subarrays requires $O(N^2)$ time.

Cow Checkups - Implementation

- Can be found at https://github.com/VihaanAnand/Coding-Vihaan/blob/main/usaco-2025-01/bronze/3.cpp

Coding w/ Vihaan

4.3B subscribers

Subscribe

65.6K

Share

16.8M views 1 second ago