CodeNection 2024 Preliminary Round Editorial

Competition Team of CodeNection 2024

Contents

1 Introduction

The rounds are prepared by the CodeNection 2024 Competition Team, with **special thanks to Wong Yen Hong, MMU** for his invaluable support and advice, which were crucial in shaping the problem sets. The team members are:

- Lim Xin Yee, MMU
- Koay Yee Shuen, MMU
- Ng Kai Xuen, MMU
- Jordan Ling Shen Hang, MMU
- Chong Jian Sieuang, MMU
- Izzminhal Akmal Bin Norhisyam, MMU

The first seven problems in this editorial belong to the closed category, while the last seven belong to the open category.

The source codes for the solutions are written in C++ and they can be found in this GitHub repository:

https://github.com/xinyee1109/codenection-2024

Lastly, we would like to express our gratitude to the testers for testing the round and providing critical comments:

- cabbitstherat, HKUST
- Loh Kwong Weng, NUS
- Vee Hua Zhi, NUS
- Ephraim Tee, MMU
- Tan Dong Yu, MMU

2 Editorial

2.1 Codey and CodeNection

Solution

For each row i from 0 to $n-1$:

- Output $(n-i-1)$ spaces for the right alignment
- Output the first $i+1$ letters from the string

Time Complexity: $O(n^2)$

2.2 Codey and Pebbles

Inspired by: Codeforces Beta Round 25 - IQ Test

Solution

First, we count the number of odd and even numbers in the array, and also storing the index of the last odd number and the last even number encountered.

- If the number of even numbers are more than odd, output the index of the last odd number.
- Otherwise, output the index of the last even number.

Time Complexity: $O(n)$

2.3 Codey and Spam

Inspired by: Educational Codeforces Round 117 - Chat Ban

Solution

To determine how many lines Codey can spam, we can represent the number of `CODENECTION` strings in each line as an array. This array starts with numbers increasing from 1 to n and then decreases symmetrically from $n-1$ back down to 1.

In the most straightforward approach, we iterate through this array, summing up the numbers in each line. At each step, we check if the total has reached or exceeded x. If it has, the current line number is the maximum number of lines that Codey can spam.

For a more efficient solution, we may use binary search to quickly find the line where the sum of strings equals or exceeds x. We know that the total number of lines is $(2 \cdot n - 1)$, with the number of strings increasing from 1 to n and then decreasing symmetrically.

- For line number $k \leq n$, the total number of strings from line 1 to line k is the sum of the first k natural numbers:

$$\frac{k \cdot (k+1)}{2}$$

- For line number $k > n$, we can first get the sum of the strings in all $(2 \cdot n - 1)$ lines and subtract it with the sum of strings from line $(k+1)$ down to line $(2 \cdot n - 1)$.

If the sum of strings found in line k is equal to x, output the line number. Otherwise, we will adjust our search range and continue looking for a match.

Time Complexity: $O(n)$ or $O(\log n)$

2.4 Codey and Coins

Inspired by: Codeforces Round 312 - Lala Land and Apple Trees

Solution

We begin by counting the number of money bags on the left ($x_i < 0$) and right ($x_i > 0$). Then, we sort all the money bags in ascending order of their positions.

To simulate Codey's alternating collection pattern, two pointers are initialized:

- one pointing to the nearest money bag on the left,
- and the other pointer pointing to the nearest money bag on the right.

In each iteration, Codey collects coins from the closest bag on both sides, alternating directions. After collecting coins, the pointers are moved further outward to the next closest bags on each side.

If one side has more money bags than the other, Codey collects coins from the extra bag on that side after balancing both sides as much as possible. This ensures maximum coins are collected efficiently.

Time Complexity: $O(n \log n)$

2.5 Codey and Rectangles

Inspired by: Codeforces Round 952 - Secret Box

Solution

To solve this problem, we start by iterating over all possible widths for the rectangle.

For each i from 1 to x, we check if it can form a valid rectangle by:

- Verify if k is divisible by i. If it is, calculate the corresponding height b as:
$$b = \frac{k}{i}$$
- Check if this height b fits within the height of the paper, y.

If both conditions are satisfied, the rectangle dimensions are valid. For valid dimensions, we can then calculate the number of unique positions where it can be placed. This can be determined by the number of horizontal positions multiplied by the number of vertical positions:

$$(x - i + 1) \cdot (y - b + 1)$$

After iterating through every possible dimensions, we can output the maximum number of unique positions where the rectangle can be drawn on the paper.

Time Complexity: $O(n)$

2.6 Codey and Manuscript

Inspired by: Codeforces Round 964 - Slavic's Exam

Solution

We can solve this problem using a two-pointer approach. The first pointer, i tracks the current character in string s, while the second pointer, j tracks the current character in string v.

For each character in string s:

- If the character matches v_i or if the character in string s is $*$, move j forward.
- Always move i forward, as we are scanning through s.

After the traversal, if pointer j reaches the end of string v, string v is a valid subsequence of string s. Otherwise, is it not a valid subsequence.

Time Complexity: $O(n)$

2.7 Codey and Recipe

Inspired by: Codeforces Round 938 - Inaccurate Subsequence Search

Solution

We can approach this problem efficiently using sliding window technique combined with map to keep track of the frequency for each type of ingredients that appear in both the kitchen and the recipe.

We uses two maps:

- Recipe map b. This map count how many times each ingredient appears in Zoey's recipe t.
- Window map c. It will count how many times each ingredient appears in the current window of size m in t.

Also, we use a counter *match* to trace how many of the ingredients in the current window match the ones in t.

1. Initial Window
   We start by looking at the first window of size m. For each ingredient a_i,
   - We increment *match* if $b[a_i] > c[a_i]$. Otherwise, we do nothing because we already have enough of ingredient type a_i. Then, we increment $c[a_i]$.

   Finally, if the *match* is greater than or equal to p, this subarray is valid, so we count it.
2. Sliding Window
   We then "slide" the window to the right until it hits the end. For each window that ends at i,

- If $c[a_{i-m}] \leq b[a_{i-m}]$, we decrement *match*. Otherwise, we do nothing because we have an excess of ingredient type a_{i-m}. Then, We decrement $c[a_{i-m}]$ because it's no longer in the window.
- For the new ingredient a_i , if $c[a_i] < b[a_i]$, then, we increment *match*. Then, we increment $c[a_i]$.
- If the new *match* value is greater than or equal to p, we count this new subarray as valid.

After sliding the window across all possible subarrays of length m, we output the total number of valid subarrays.

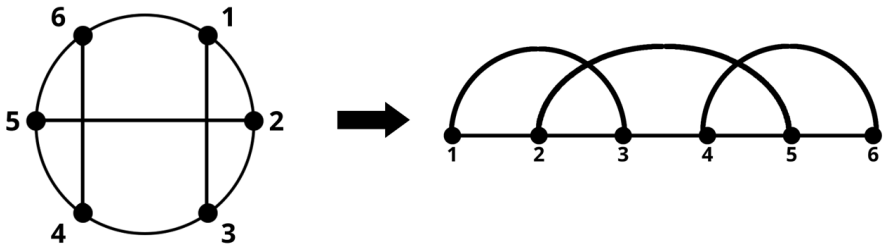
Time Complexity: $O(n + m)$

2.8 Codey and Toy Kingdom

Inspired By: AtCoder ABC 338 - Chords

Solution

The main observation for this problem is that we can actually split open the circle between 1 and $2n$ into a one dimensional axis without any loss of information:



Now, this problem is simply about finding the total pairs of segments that has an intersection.

To solve this, we iterate through each point from 1 to $2n$ to process the segments while maintaining an array s of length $2n$. This array keeps track the **current** number of **open** segments that have an ending point at j, denoted by s_j. Specifically, at each point i:

- We use s to keep track of the ending points of the segments that start before i and end after i (open segments).
- For each segment that starts at i, we increment s_j by one, where j is the segment's ending point. The total number of segments that intersects with the current segment can be found using $\sum_{k=i+1}^{j-1} s_k$, i.e., the number of open segments with ending points between the starting and ending points of the current segment.

- We set $s_i = 0$, because we have finished processing the segments before i.

The final answer is the sum of the intersections found using the process described above.

Naively computing the range summation in s would result in an $O(n^2)$ solution. Hence, we use Segment Tree or Fenwick Tree to compute the range sum efficiently.

Time Complexity: $O(n \log n)$

2.9 Codey and Peaks

Authored by: Wong Yen Hong

Solution

We will focus on finding the number of combinations for the interval between the two peaks x and y, with height a_x and a_y, respectively. The solution discussed here can also be applied to both ends of the array by assuming the height of the ends to be $10^5 + 1$.

First, it's not hard to see that the interval should satisfy the following pattern:

$$a_x > a_{x+1} \geq a_{x+1} \geq ... \geq m \leq ... \leq a_{y-2} \leq a_{y-1} < a_y$$

where m is the minimum height of the interval.

To count the number of combinations between the peaks x and y, we can model it as a stars and bars combinatorics problem by fixing the minimum, m where $1 \leq m \leq min(a_x - 1, a_y - 1)$.

For every m, we can think of it as the total number of ways to distribute $n - 1$ numbers into these boxes:

$$a_x - 1 \mid a_x - 2 \mid ... \mid m + 1 \mid m \mid m + 1 \mid ... \mid a_y - 2 \mid a_y - 1$$

Here, we have $(a_x - m + 1) + (a_y - m) - 1$ bars and $n - 1$ stars. One of the stars is fixed at m, which acts as a separator between the non-increasing and the non-decreasing parts of the bars, avoiding double-counting. This yields the following equation:

$$\sum_{i=1}^{m} {}^{((a_x-i+1)+(a_y-m)-1)+(n-1)}\mathrm{C}_{n-1}$$

Finally, we simply need to combine the answer for the three intervals by getting their products.

Time Complexity: $O(n \log n)$