CodeNection 2024 Final 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 Exit

Solution

Output the sum of absolute differences of the x-coordinates and y-coordinates.

Time Complexity: $O(1)$

2.2 Codey and Gardening

Inspired by: Codeforces Beta Round 2

Solution

To solve this problem, we can first sort all participants by their total score in descending order. As we iterate through the sorted list of participants, we assign ranks:

- If a participant has a different score from the previous one, we increment the rank.
- Otherwise, they will share the same rank.

Finally, we can just output Codey's ranking.

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

2.3 Codey and Symbol

Authored By: Wong Yen Hong

Solution

If we try to calculate these values directly, we could encounter floating-point errors because computers cannot capture the absolute precision due to the infinite number of decimals.

Hence, we can rearrange the expressions algebraically to compare them in terms of integer instead, which is precise:

$$a - \frac{b}{c} \; ? \; \frac{d}{e} - f$$

$$\frac{(a \cdot c - b)}{c} \; ? \; \frac{(d - f \cdot e)}{e}$$

$$(a \cdot c - b) \cdot e \; ? \; (d - f \cdot e) \cdot c$$

Then, we can just compare them and output their relationship.

Time Complexity: $O(1)$

2.4 Codey and Rectangles 2

Inspired by: AtCoder ABC 318 - Overlapping sheets

Solution

The main idea is to track each grid square that is covered by at least one sheet. To do this, a 2D array can be used where each cell represents a $(1 \cdot 1)$ grid on the floor. We mark a cell in this 2D array as "covered" (1) if it's covered by any rectangle.

Finally, we sum up all the "covered" cells to calculate the total area.

Time Complexity: $O(n \cdot \max(b_i - a_i) \cdot \max(d_i - c_i))$

2.5 Codey and Jutsu

Inspired by: Codeforces Round 706 - Diamond Miner

Solution

First, since we know that the signs of the coordinates do not matter, we will assume all coordinates are non-negative. Second, since all the clones and all the aliens are positioned on the y-axis and the x-axis respectively, the distance between y_j and x_i can be defined as:

$$\sqrt{y_j^2 + x_i^2}$$

To minimize the total sum of distances between clones and aliens, we sort the clones' y-coordinates and aliens' x-coordinates in ascending order. Then, we can pair the i-th clone, y_i with the i-th alien and it will result in the minimum sum of distances:

$$\sum_{i=1}^{n} \sqrt{x_i^2 + y_i^2}$$

To see why this works, we can look at how assigning a clone to different aliens affect the distance. Given a clone y_i, and two aliens, x_j and x_k where $x_j > x_k$, the following equation will always hold:

$$\sqrt{y_i^2 + x_j^2} - \sqrt{x_j^2} < \sqrt{y_i^2 + x_k^2} - \sqrt{x_k^2}$$

Meaning that, pairing a clone y_i with an alien with bigger x_j will always result in a smaller increase in overall distance compared to pairing it with a smaller x_k. This is because $\sqrt{x}$ is a monotonic function that has a faster

growth rate for smaller x compared to bigger x.

Thus, we can use a greedy approach. Starting from the largest y_i, we will minimize the increase in overall distance by pairing y_i with the largest x_j that has not yet been paired. Doing this is equivalent to the algorithm described in the second paragraph.

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

2.6 Codey and Toy Kingdom 2

Inspired By: AtCoder ABC 288 - Don't be cycle

Solution

The key of solving this problem is knowing that the minimum number of edges required for an undirected graph of N nodes to be fully connected is $N - 1$. This essentially forms a tree. If the number of edges is fewer than $N - 1$, the graph will not be connected. If the number of edges is bigger than $N - 1$, then the graph will have cycles (redundant edges).

Let's focus on the first subproblem first – finding the number of edges needed to be destroyed. To do that, we first need to locate each connected component. For each connected component i, we must know, $nodes_i$ and number of edges, $edges_i$ in that component. Using the earlier observation, we can calculate the number of edges to be destroyed using the following formula:

$$destroy_i = edges_i - (nodes_i - 1)$$

The answer to the first subproblem is simply the sum of $destroy_i$ for each connected component i.

For the second subproblem – finding the number of edges needed to be added for the graph to be connected. To solve this, we can treat each connected component as a node in the graph. Based on the same observation, the minimum number of edges required to connect all these components is simply the number of connected components - 1.

Time Complexity: $O(n)$

2.7 Codey and Speeches

Inspired By: Codeforces Round 723 - Potions (Hard Version)

Solution

We will process Codey's friends from left to right while maintaining all the friend that Codey have gave speeches in a data structure like a priority queue or a multi-set. These data structures keep all elements sorted and support insertion and removal in $O(\log n)$. For each friend i, if we can give speeches without depleting our total feedback score below 0, we will give speech to that friend. Otherwise, if a_i is greater than the most negative feedback score we have received from a friend, we will swap the two feedback scores. We can easily find the most negative feedback score using the data structure.

To see why this works, we will show it by induction. At each point i (for the first i friends), we define the optimal state at point i as (k, b) where k denotes the maximum number of friend we can give speeches to at i and b denotes the maximum feedback score we can have after giving k speeches at i.

At $i = 1$, it can be easily seen that using the approach above will ensure the optimal state. Assuming the optimal state remains at point $i = N$, at point $i = N + 1$, we will either:

- Give speeches to friend $N + 1$, if $b_N + a_{N+1} \geq 0$
- Swap out the most negative feedback score from a friend if a_{N+1} is greater than it
- Do nothing

Notice that, regardless of what we do, the state of $N + 1$ will not worsen compared to the state of N; it will only improve or maintain the previous state. If we follow the approach above, it guarantees that we achieve the optimal state at $i = N+1$. Hence, this proves the optimality of the approach above.

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

2.8 Codey and Zoey 2

Inspired by: AtCoder ABC 274 - Robot Arms 2

Solution

This problem can be reduced down to finding whether it is possible to satisfy the following equations by changing the signs of each number (except a_0):

$$x = a_1 \pm a_3 \pm ... \pm a_{2 \cdot \lceil n/2 \rceil}$$
$$y = \pm a_2 \pm a_4 \pm ... \pm a_{2 \cdot \lfloor n/2 \rfloor}$$

This is because all odd-indexed a_i moves in the horizontal axis while all the even-indexed a_i moves in the vertical axis.

Now, we solve it using DP (Dynamic Programming). Without the loss of generality, we will only focus on solving the horizontal axis, as vertical axis can be solved similarly. We denote $dp_h[i][j]$ as whether it is possible to reach j on the horizontal axis using the first i moves. The DP transition is defined as:

If $dp_h[i][j]$ is True, then:

$$dp_h[i+1][j+a_i] = \text{True}$$
$$dp_h[i+1][j-a_i] = \text{True}$$

For horizontal axis, we initialize $dp_h[1][a_1]$ as True, because Codey started at $(a_1, 0)$. While for vertical axis, we initialize $dp_v[1][a_1]$ and $dp_v[1][-a_1]$ as True.

Directly using a two dimensional array to represent dp states may not be doable. Hence, we can offset the coordinates by the maximum coordinate.

The final answer is `YES` if both $dp_h[\lceil \frac{n}{2} \rceil][x]$ and $dp_v[\lfloor \frac{n}{2} \rfloor][y]$ are True.

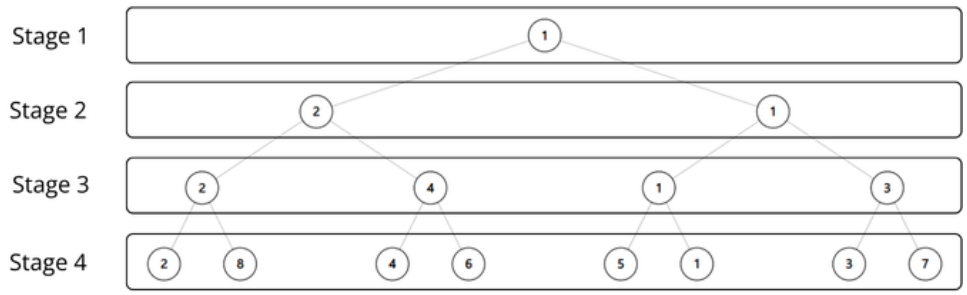
Time Complexity: $O(n^2)$

2.9 Codey and NectionCode

Inspired by: Educational Codeforces Round 149 - Playoff Fixing

Solution

For convenience, in this editorial, we will denote the final stage as stage 1 and the first stage as stage n. This is illustrated in the figure below:



To achieve the desired ranking, the first observation is that team i must only lose at stage $\lceil \log_2(i) \rceil + 1$. The second observation is at stage j, each losing team i, where $\lceil \log_2(i) \rceil + 1 = j$ should not face off with another losing team, meaning that, each team i should face off with another team k, where $\lceil \log_2(k) \rceil + 1 > j$.

First, we assume none of the teams has a fixed seed. To count the total number of ways to achieve the desired ranking, we calculate the total ways to arrange the losing teams at each stage, from stage n to stage 1 and then combine them.

Let us begin by counting the number of ways to arrange the losing teams at any stage i. At stage i, there are m_i matches ,where $m_i = 2^{i-1}$ and m_i teams are eliminated. The total number of ways to arrange all the m_i teams to be eliminated is:

$$^{m_i}\mathrm{P}_{m_i} \cdot 2^{m_i}$$

This accounts for $^{m_1}\mathrm{P}_{m_i}$, the number of ways to permute the m_i teams in the m_i matches, and 2^{m_i}, which accounts for for the two possible arrangements for each match (team A vs team B, team B vs team A).

Repeating this process for every stage, the answer is the product of the number of ways to arrange the losing teams at each stage i.

Now, let us introduce teams with fixed seeds. The concept remains mostly the same, but there are scenarios where the fixed seeds make it impossible to achieve the desired ranking. Specifically, this happens when a match involves two teams with fixed seeds, and both must either lose or win that match.

Additionally, at any stage i, if there are p_i teams with fixed seeds that must lose at the stage and q_i matches that include at least one team with fixed seed. Then the total number of ways to arrange the losing teams in that stage becomes:

$$^{m_i-p_i}\mathrm{P}_{m_i-p_i} \cdot 2^{m_i-q_i}$$

This is because $m_i - p_i$ represents the teams without fixed seeds, which can be permuted freely, and $m_i - q_i$ represents the matches without fixed seeds, which can be rearranged (Fixed-seed matches cannot be rearranged).

This yields the final equation:

$$\prod_{i=1}^{n} {}^{m_i-p_i}\mathrm{P}_{m_i-p_i} \cdot 2^{m_i-q_i}$$

To find p_i and q_i for any stage, we can simulate it starting from stage n to

stage 1.

Time Complexity: $O(2^n + n \log n)$