CodeNection 2023 Preliminary Round Editorial

Competition Team of CodeNection 2023

Contents

1 Introduction

The rounds are prepared by the Competition Team of CodeNection 2023:

- Wong Yen Hong, MMU
- Tung Tze Yang, MMU
- Marcus Chin Wei Hern, MMU
- Kalla Deveshwara Rao A/L Rama Ra, MMU
- Law Chin Keat, MMU
- Chan Kar Kin, 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 solutions are written in C++ and they can be found in this GitHub repository:

`codenection-2023` - https://github.com/wyhong3103/codenection-2023

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

- rabbitsthecat, HKUST
- Geremie Yeo
- Melody Koh Si Jie, MMU
- Tham Joe Ming, MMU
- Lim Xin Yee, MMU

2 Editorial

2.1 Codey and CodeNection

Solution

Print `I LOVE CODENECTION` for n times.

Time Complexity: $O(n)$

2.2 Codey and Hide-and-Seek

Solution

Find `#` and print its coordinate, otherwise, print -1.

Time Complexity: $O(nm)$

2.3 Codey and Math

Solution

It turns out it is always possible to get n using l and r.

When $n = 0$, we can simply set $l = 0$ and $r = 0$.

Notice that when $n \neq 0$, we can always select $l = -(n-1), r = n$ and we will always get a sum of n, which can be shown by this equation below:

$$\left(\sum_{i=-(n-1)}^{n-1} i \right) + n = 0 + n = n$$

Time Complexity: $O(1)$

2.4 Codey and Textbook

Inspired by: Codeforces Round 304 (Div. 2) - Soldier and Bananas

Solution

We can easily find the *cost* using a loop or the sum of natural numbers formula, and then print $max(0, cost - k)$.

Time Complexity: $O(n)$ or $O(1)$

2.5 Codey and Money

Solution

First, it's important to note that a ringgit bill of 10^i can contribute to 10^{i+1} but not the reverse. For example, ten 1 ringgit bills are equivalent to one 10 ringgit bill.

This allows us to greedily pick the bills from the smallest to the greatest. For each 10^i ringgit bill, we will first check if the current number of 10^i bills we have is greater than or equal to the corresponding digit in n. If it's not, that means we cannot form n because no larger bills can help us reach n. After each check, we will subtract the current number of bills by the corresponding digit of n. If there are j remaining bills of the 10^i ringgit denomination, we will add $\lfloor \frac{j}{10} \rfloor$ to the number of 10^{i+1} ringgit bills we have. We will repeat the same process until either n is formed or it fails to form.

It is not possible to store n as a number in most programming languages, so we will store it as a string.

Time Complexity: $O(k)$

2.6 Codey and Team Selection

Solution

To maximize the total skill of the team, we need to maximize the additional skill points for each member.

Notice that, for each member with an index i, it is not possible to get additional gain s_j where $j > i$. Hence, it becomes clear that member i can only get s_j where $j \leq i$.

Let's show that for each member i, there is a way to get $max(s_j)$ where $j \leq i$, which is also known as prefix maximum. The key idea is to select team members starting from the first occurrence of the largest prefix maximum of s_j and continue this process until all members with indices greater than or equal to j are included in the team. Subsequently, we move on to the first occurrence of the second-largest prefix maximum of s_j and repeat the process, and so forth, until all members are selected.

Time Complexity: $O(n)$

Bonus

The original version of this problem includes an additional constraint: During the selection process, you can choose to remove one member from the array of unselected members. This means that your team's skill level remains the same after removing that member, and you can perform this step at most m times. The values of $-10^5 \leq a_i$ and $s_i \leq 10^5$ apply in this context.

Bonus Solution

Each member has an optimal value of s_j, which represents their prefix maximum. Initially, we can solve the first version of the problem, and then subtract the answer by the sum of the smallest m (or fewer) negative values after including the optimal gains.

To understand why this approach works, consider a scenario where we aim to carry out the selection process in such a way that each member achieves their optimal s_j. In this case, our objective is to remove member i such that it doesn't affect the optimal gain for the other elements. We can achieve this by removing member i at the same step where it is initially selected because it will be removed anyway in that step. Consequently, the optimal gain for each element remains unchanged even after their removal. This same logic can be applied to all members that are going to be removed.

2.7 Codey and Painted Tree

Solution

* To prevent any ambiguity, we will use the variable N to represent the total number of nodes in the tree given, while the variable n will be used in its general combinatorial context.

First, it is important to know that for every pair of nodes (u, v) in a tree, there is only one unique path from u to v, and vice versa. As a result, the number of simple paths in a tree with n nodes is $\binom{n}{2}$ since there exist exactly $\binom{n}{2}$ pairs of nodes (u, v) where (u, v) and (v, u) are considered equivalent.

With this fact in mind, we can deduce that the number of black nodes, b must satisfy the following condition:

$$\binom{b}{2} = m$$

We can find the number b by manually iterating each integer i from 1 to n, b is found if and only if $\binom{i}{2} = m$.

Note that we won't always be able to find a value of b that satisfies the condition because not every number m is a valid value for $\binom{n}{2}$. However, if we are able to determine the value of b, the problem is then reduced to determining the number of ways to color b nodes black in a tree with N nodes, because each way of coloring b black nodes in the tree contributes m unique b-paths, so we need to find how many ways to color b nodes in the tree. In this case, the answer is simply given by $\binom{N}{b}$.

One thing to note is that when $m = 0$, the answer should be 1.

Finally, directly calculating the value of $\binom{N}{b}$ and then taking the modulo with $10^9 + 7$ may not be feasible in some programming languages due to the large integers involved. To address this, we need to use modular arithmetic to compute the answer, more about it can be found in this Codeforces blog, Modular Arithmetic for Beginners by Spheniscine.

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

2.8 Codey and Number Grid

Inspired By: TheForces Round 8 - Permutation Forces

Solution

To solve this problem, we need to use DSU (Disjoint Set Union) to merge connected components and check the size of a connected component that consists of node i. Initially, each number on the grid is a component of its own with a size of 1. We can iterate from 1 to $n \cdot m$. For each iteration i, we merge the component that consists of number i with the component of the adjacent numbers j if and only if $j < i$. Additionally, we also check if the size of the component that consists of 1 is equal to i. If it is, that means the first i numbers are all in the same component, and we find the maximum of such i.

Time Complexity: $O(nm \cdot \alpha(nm))$

2.9 Codey and Crimes

Inspired by: Codeforces Beta Round 19 - Checkout Assistant

Solution

Let's consider DP (Dynamic Programming).

We define the DP state, $dp[i][j]$, as the minimal amount of ringgits to get the first i items while having j seconds to steal.

The DP transition is defined as follows:

If we buy item i,

$$dp[i][j + a[i]] = min(dp[i][j + a[i]], dp[i - 1][j] + b[i])$$

If we steal item i,

$$dp[i][j - 1] = min(dp[i][j - 1], dp[i - 1][j])$$

Note that in order for this to work, we need to allow j to be negative during the transition. This is to ensure that we are not limited by the order. In fact, the only thing that matters in this problem is finding the optimal set s of items we're buying, once we find this set, we can always guarantee to use a greedy order to get all n items. The greedy order of buying and stealing is we can first buy the items we chose and then steal the rest. This approach ensures that the time we have does not go negative at any instant.

While the state allows for negative j, the final answer has to be $dp[n][j]$ where $0 \leq j$. The reason is the total seconds it takes to process cannot be negative at the end, otherwise, we cannot afford to steal the items.

We need to be careful with the implementation of the DP transition to ensure it runs in $O(n^2)$. Notice that for the first transition, $dp[i][j + a[i]]$, we can set an upper bound of n for the second dimension to ensure it does not exceed n since we don't need more than n seconds to steal items.

Finally, we cannot use a negative index to access an array. To address this, we can represent negative numbers in a non-negative form, which is equivalent to applying an offset to the number.

Time Complexity: $O(n^2)$