2023 Pacific Northwest Division 2 Solutions

The Judges

Feb 24, 2024

A Cappella Recording

Problem

- You are given n integers. In a single operation, you can pick a collection of integers and remove all of them, as long as the difference between the maximum and minimum integer in the collection is at most d. Compute the minimum number of operations needed to remove all n integers.

Initial Observations

- You will need to remove the smallest integer at some point.
- When you do so, you should remove as many integers as possible.

A Cappella Recording

Solution

- Let x be the smallest integer in the list. It is optimal to remove all integers in $[x, x + d]$.
- Naively simulating this process is too slow. However, if we sort the list, we can keep track of the current minimum and loop over the integers in nondecreasing order.
- This runs in $\mathcal{O}(n \log n)$.

Champernowne Verification

Problem

- Determine if an integer n is composed of the first k integers written in increasing order.

Solution

- There are multiple possible approaches. For example, given the length of n, generate the integer $123 \ldots k$ and see if it matches.
- Since there are only nine valid entries, one could also explicitly handle all entries with nine if statements.

Four Die Rolls

Problem

- You are given n integers. Determine the number of arrays of size 4 where the first n integers match exactly, all integers in the array are in the range $[1, 6]$, and all integers are distinct.

Solution

- There are only 6^4 possible arrays, so we can check all of them with brute force.
- There is also a closed-form solution based on whether all n integers are already distinct.

Ordered Problem Set

Problem

- You are given a permutation of the first n positive integers. Compute all values of k such that k divides n and the first k integers in the array are a permutation of 1 through k, the next k integers are a permutation of $k+1$ through $2k$, and so on.

Solution

- Because n is small, we can check this with brute force for all possible values of k.
- Care should be exercised to confirm that k is divisible by n.

Training

Problem

- Given a starting integer s and n intervals $[l_i, r_i]$, if your integer currently falls inside the interval, you can increment it or leave it unchanged. Compute the maximum possible value your integer can end up as.

Solution

- It is always optimal to increment the integer when given the chance. We leave the proof of this as an exercise to the reader.
- Therefore, we read in the intervals one by one and increment if the integer falls in the given range.

Triple Sevens

Problem

- Given three sets of digits, determine if every set of digits contains a 7.

Solution

- Due to the structure of the problem, there are multiple approaches to check whether every set has a 7.
- For example, one could parse the integers in each line and look for a 7.
- Alternatively, one could read three lines and check if each line has 7 as a substring.

ABC String

Problem

- You are given a string with n `A`'s, `B`'s, and `C`'s. Partition the string into the minimum number of subsequences such that each subsequence is the concatenation of several permutations of `ABC`, not necessarily the same permutation in each instance.

Initial Observations

- Note that if the first three letters of a string are some permutation of `ABC`, then if we partition the remainder of the string optimally, we can always prepend the first three characters to an arbitrary subsequence.
- Therefore, we can greedily assign characters to subsequences looping over the characters in order.

ABC String

Solution

- If there is an active subsequence of length 2 that does not contain the character we are currently considering, assign the current character to that subsequence and set that subsequence to be empty.
- Otherwise, if there is some subsequence of length 1 that does not contain the character we are currently considering, assign the current character to that subsequence.
- Otherwise, create a new subsequence consisting of just that character.
- Since we always reset the subsequence lengths when they reach 3, this algorithm runs in $\mathcal{O}(|s|)$.

Acceptable Seating Arrangements

Problem

- You are given a grid of integers of size at most 20×20 where each row is sorted in increasing order. In a single operation, you can pick two elements and swap them, as long as all rows are still sorted. Given a final configuration where all rows are sorted, get from the initial configuration to the final configuration in at most 10^4 operations.

Initial Observations

- Let $r = c = 20$, and note that $20^3 < 10^4$. If we can place a single element in the correct location in at most c operations, we can place all elements in $rc^2 < 10^4$ operations.
- We therefore try to place the elements in the correct locations in increasing order of value.

Acceptable Seating Arrangements

Solution

- Let the current element we are trying to place correctly be x. Inductively, all elements less than x are correctly placed.
- If x is in the same row in both configurations, then x must be correctly placed already, by the induction hypothesis.
- Otherwise, by the induction hypothesis, x must go to the leftmost column in the row of the final configuration that is not already filled, let that element be called y.
- The only scenario where that is not directly possible is if y is greater than the element currently to the right of x.
- However, in that case, we can swap y with the largest element in the row where x currently is that is less than y.
- We repeat the above until x can be placed in the correct place. This must happen after at most c operations.

Balanced Tree Path

Problem

- You are given a tree where each node has an opening or closing bracket, brace, or parenthesis. A path along the tree generates a string by concatenating all characters on the path. The string is balanced if the characters match up properly. Compute the number of paths that yield balanced strings.

Initial Observations

- There are $\mathcal{O}(n^2)$ distinct paths in the tree, so if we can check them all in $\mathcal{O}(1)$ time, that would run in time.
- Naively, it takes $\mathcal{O}(n)$ time to check if a string is balanced, but due to the nature of the paths, a lot of redundant work is performed.

Balanced Tree Path

Solution

- We can DFS through the tree, maintaining a stack of characters seen.
- If the current character is a closing character, make sure it matches properly, otherwise terminate the recursion. If it matches properly, pop the character from the stack.
- Otherwise, the current character is an opening character and should be pushed on to the stack.
- If the stack is empty, we have observed a balanced path.
- This optimizes the $\mathcal{O}(n^3)$ solution to $\mathcal{O}(n^2)$.

Candy Factory

Problem

- You have n factories that each produce some number of candies. A bag of candies is valid if it contains exactly k candies, and no two candies came from the same factory. Each factory has already produced some candies, and you wish to bag all the candies such that all bags are valid. You can order any factory to produce additional candies. Compute the minimum number of additional candies over all factories that need to be produced such taht all produced candies can be put into valid bags.

Initial Observations

- If it is possible to produce additional candies such that exactly b valid bags can be formed, it is also possible to produce additional candies such that exactly $b + 1$ valid bags can be formed. Therefore, we can binary search for the answer.

Candy Factory

Solution

- We wish to identify if b bags can be formed.
- Two clearly necessary conditions are that $b \cdot k$ must be greater than or equal to the number of candies produced, and that no factory produced more than b candies.
- We leave as an exercise to the reader the proof that these two conditions are sufficient.
- This algorithm runs in $\mathcal{O}(n \log n)$.

Alternate Solutions

- Because n and k were small, a fast simulation based on the above that ran in $\mathcal{O}(nk \log n)$ could pass.
- It is also possible to derive a closed-form answer using the above observations.

Item Selection

Problem

- You are given a UI with some items initially selected. You can click between pages, toggle individual selections, select all items on a page, or deselect all items on a page. Compute the minimum number of clicks needed to get to a final configuration of items selected.

Initial Observations

- In terms of moving between pages, it is never optimal to change direction more than once. Therefore, one should either move left and then move right, or move right and then move left. It is possible that the movements are empty.
- On a given page, we should only use at most one of the select all and deselect all options.

Item Selection

Solution

- For each page, we do some casework to compute the minimum number of button clicks based on whether we use select all, deselect all, or neither.
- We then keep track of all pages that require changes. We compute the minimum number of clicks needed to navigate pages by either going left and then going right, or going right and then going left.

Magic Cube

Problem

- You are given an $n \times n \times n$ cube. Support rotating contiguous layers of the cube and point queries for what cube is at a given location.

Initial Observations

- n is too large to store the entire cube in memory.
- However, n and q are relatively small, and most of the cube does not matter.

Magic Cube

Solution

- We are going to consider the reverse process.
- Specifically, given a query point, we can consider the rotations in reverse order to see which point is being queried in the initial cube.
- For each rotation, we can see if the query point would have been impacted by the rotation, and if so, rotate the query point the other way.
- We can then compute, based on the initial point, which cube it is.

Sequence Guessing

Problem

- You are asked to generate a sorted sequence of integers starting with 0 ending with 10^5 where adjacent entries differ by either 1 or 2. You tell an adversary how long your sequence is, then the adversary will guess integers in $[0, 10^5]$ and you must tell the adversary either the index of that integer in your list or assert that it is not present. Generate and maintain such a sequence while forcing the adversary to guess an integer that is not present 33333 times.

Initial Observations

- $\left\lfloor \frac{10^5}{3} \right\rfloor = 33333.$
- If you always include every multiple of 3, then there are 33333 pairs of adjacent integers and you can always force the adversary to miss by selecting the other integer in the pair.

Sequence Guessing

Solution

- The sequence will take the form $0, a_2, 3, a_4, \ldots, 99996, a_{66666}, 99999, 10^5$.
- Tell the adversary that your sequence has length 66668.
- If the adversary guesses $3x + 1$, and the adversary has not guessed $3x + 2$, report it is not present and include $3x + 2$ in your sequence.
- Otherwise, if the adversary guesses $3x + 2$, and the adversary has not guessed $3x + 1$, report it is not present and include $3x + 2$ in your sequence.
- Otherwise, the entry must already be in your sequence, report its index accurately.