Solution: Find the Corrupt Pair

Let's solve the Find the Corrupt Pair problem using the Cyclic Sort pattern.

We'll cover the following

Statement

We are given an unsorted array, `nums`, with n elements and each element is in the range $[1, n]$ inclusive. The array originally contained all the elements from 1 to n but due to a data error, one of the numbers is duplicated, which causes another number missing. Find and return the corrupt pair (missing, duplicated).

Constraints:

- $1 \leq n \leq 10^3$
- $1 \leq$ `nums[i]` $\leq n$

Solution

The given input array is unsorted, and the numbers in the input array lie in the range 1 to n. Therefore, the cyclic sort is optimal to place the elements at their correct index. Once all the elements are in their correct position, we can easily find the pair of missing and duplicate numbers.

The steps of the above-mentioned approach are given below:

1. The first step is to place each number in its correct position during the cyclic sort. Initialize `i` to 0 and iterate over the array. While traversing, we will perform the following steps:
   - The variable `correct` is calculated as `nums[i] - 1`, representing the index where the current number should be placed if the array were sorted.
   - If the current number, `nums[i]`, is not equal to the number at the correct position, `nums[correct]`, a swap is performed using the `swap` function to move the current number to its correct position.
   - If the numbers are already in their correct positions, `i` is incremented to move to the next element.
2. The second step is to traverse the array and detect the number that is not at its correct index. This will be the duplicated number. When we add 1 to that index, the resulting value will be our missing number.
3. Return the pair containing missing and duplicated numbers.

Let's look at the following illustration to get a better understanding of the solution:

We have an unsorted array. Let's apply cyclic sort to find the corrupt pair. We'll keep elements at their correct index.

1 of 10

Java

```java
import java.util.*;

class FindCorruptPair {
    public static int[] findCorruptPair(int[] nums) {
        // Initialize missing and duplicated
        int duplicated = 0;
        int missing = 0;
        int[] pair = {
            0,
            0
        };
        // Apply cyclic sort on the array
        // Traversing the whole array
        int i = 0;
        while (i < nums.length) {
            // Determining what position the specific element should be at
            int correct = nums[i] - 1;
            //Check if the number is at wrong position
            if (nums[i] != nums[correct]) {
                // Swapping the number to it's correct position
            }
        }
        // Finding the corrupt pair(missing, duplicated)
        for (int j = 0; j < nums.length; j++) {
```

Find the Corrupt Pair

Time complexity

The time complexity of the above solution is $O(n)$, because it uses the cyclic sort, where n is the size of the array. In the worst case, let's assume that every element is placed at the wrong index. With every swap operation, there is at least one element that is placed in the right place. The total number of swap operations that need to be performed is $O(n)$.

Space complexity

The space complexity of the above solution is $O(1)$, because we have used constant space.

Back
Find the Corrupt Pair

Next
Find the First K Missin...

Mark as Completed