2670. Find the Distinct Difference Array

Problem Description

Easy

Hash Table

The problem presents us with an integer array called nums, indexed from 0, which means the first element of the array is at index

1. The "prefix" which includes all elements from the start of nums up to and including the element at index i. 2. The "suffix" which includes all elements after index i up to the end of the nums.

Using two sets, we can capture the unique elements in the prefixes and suffixes of nums at each index.

nums array, we have to find the count of distinct elements in two different parts of nums:

diff[i] is then defined as the number of distinct elements in the prefix minus the number of distinct elements in the suffix.

ntuition

from index i through to index j, inclusive. If i is greater than j, it means the subarray is empty.

To construct the diff array as described, we need to keep track of the distinct elements as we move through the nums array.

Note that when working with subarrays or slices of an array, nums[i, ..., j] means we are considering the elements of nums

0. Our task is to calculate a new array, diff, also of the same length as nums, based on a specific rule. For each position i in the

The algorithm includes the following steps: Initialize an extra array suf with the same length as nums plus one, to record the number of distinct elements in the suffix

part starting from each index. We add an extra space since we are including the empty subarray when i > j.

Traverse the nums array in reverse (right to left), starting from the last element, and with each element, we encounter, add it to a set to ensure only distinct elements are counted. Update the corresponding suf entry with the length of this set to reflect the current number of distinct elements in the suffix up to the current index.

Clear the set (for counting distinct elements in the prefixes) and then traverse the nums array once more, from left to right.

- With each element, we add it to the set to keep track of distinct elements and calculate diff[i] by subtracting the number of distinct elements in the suffix (obtained from suf[i + 1]) from the number of distinct elements in the prefix (length of the set at this point in the traversal).
- The solution uses a couple of Python data structures lists and sets and leverages their properties to solve the problem efficiently: 1. Lists: They are used to store the input (nums) and output (ans for the distinct difference array, and suf for tracking the number of distinct elements in suffixes).

Sets: A set is used to keep track of distinct elements because sets naturally eliminate duplicates.

The implementation follows these steps: Initialize the suf list to have n + 1 zeros, where n is the length of nums. This is done to handle the suffix count of distinct

•

•

Solution Approach

elements as well as accounting for the empty subarray at the end. Start by populating the suf array from the end of nums moving towards the beginning. For each index i, add nums[i] to a

count of distinct elements from nums[i] to the end of the array. Once the suf array is complete, reset (clear) the set to use it for counting distinct elements in prefixes. •

set (this ensures only distinct elements are counted). Then, set suf[i] to the current size of the set, which represents the

Iterate through nums from start to end. For each index i, add nums[i] to the set (again, ensuring uniqueness) and subtract

the number of distinct elements in the suffix (obtained from suf[i + 1]) from the number of distinct elements in the prefix

(the current size of the set). This gives the diff[i] value. Assign this result to ans [i], building the answer as the loop progresses. •

The key algorithmic insight here is to use two passes, one for suffixes and one for prefixes, with the help of sets to dynamically

maintain the count of distinct elements efficiently. This negates the need to re-calculate the number of distinct elements for each

index from scratch, thus improving the time complexity significantly.

Create an answer list ans initialized with zeros of length n to hold the final result.

Finally, return the ans list once the loop is complete.

for i in range(n - 1, -1, -1): s.add(nums[i]) suf[i] = len(s)s.clear() for i, x in enumerate(nums):

In the first loop, we fill the suf array with the count of distinct elements for the suffix starting from each index. In the second

loop, we use the suf array and the prefix count of distinct elements to calculate the diff for each index, which is stored in ans.

Let's illustrate the solution approach using a small example. Suppose we have the following integer array nums:

We first populate the suf array in backward fashion:

Now our suf array looks like this: [3, 3, 2, 1, 0].

Next, we'll go over nums from left to right to build our diff array:

Our final diff array is [-2, 0, 1, 3] which is the result of our algorithm.

def distinctDifferenceArray(self, nums: List[int]) -> List[int]:

Create an empty set to store unique elements

Initialize the answer array with zeros

for i, number in enumerate(nums):

// The length of the input array

int[] suffixDistinctCount = new int[length + 1];

Set<Integer> distinctNumbers = new HashSet<>();

// starting from the end of nums array

int[] result = new int[length];

for (int i = 0; i < length; ++i) {</pre>

distinctNumbers.add(nums[i]);

// Return the computed distinct difference array

for (int $i = length - 1; i >= 0; --i) {$

unique elements.add(number)

Initialize a suffix array of length 'length + 1' with zeros

Populate the suffix count array with count of unique elements from the end

Calculate the difference between the number of unique elements seen

so far and the number of unique elements that will be seen in the suffix

// Suffix array to store the number of distinct elements from index 'i' to the end

// A set to keep track of distinct numbers as we traverse the array from the end

// Populate the suffixDistinctCount array with the count of distinct numbers

Get the length of the input list

suffix count = [0] * (length + 1)

• For i = 3, nums [3] = 3. We add this to the set s, which becomes $\{3\}$, and suf [3] = 1.

ans[i] = len(s) - suf[i + 1]

s.add(x)

Example Walkthrough

Here's the main loop in the code explained:

```
nums = [4, 6, 4, 3]
 We need to calculate the diff array where diff[i] is the count of distinct elements from the start of nums up to i, minus the
 count of distinct elements from i+1 to the end of nums. Here's how we would apply the solution step by step:
• Initialize the suf list to length n+1 with zero values. n is the length of nums, so suf initially will be [0, 0, 0, 0, 0].

    Initialize an empty set s to record distinct elements as we go.
```

• For i = 2, nums [2] = 4. We add this to the set s, which becomes $\{3, 4\}$, and suf [2] = 2. • For i = 1, nums [1] = 6. We add this to the set s, which becomes $\{3, 4, 6\}$, and suf [1] = 3. • For i = 0, nums[0] = 4. The set s is already {3, 4, 6}, so adding 4 doesn't change it, and suf[0] remains at 3.

• For i = 2, we add nums[2] = 4 to s, which doesn't change it as 4 is already in the set. Thus, we calculate diff[2] = len(s) - suf[3] = 2 -

• Clear the set s.

1 = 1.

Python

class Solution:

from typing import List

length = len(nums)

unique_elements = set()

unique elements.clear()

answer = [0] * length

Return the answer array

int length = nums.length;

return answer

• For i = 0, we add nums[0] = 4 to s, which becomes $\{4\}$. We calculate diff[0] = len(s) - suf[1] = 1 - 3 = -2.

• For i = 3, we add nums[3] = 3 to s, which becomes $\{3, 4, 6\}$. We calculate diff[3] = len(s) - suf[4] = 3 - 0 = 3.

• For i = 1, we add nums[1] = 6 to s, which becomes $\{4, 6\}$. We calculate diff[1] = len(s) - suf[2] = 2 - 2 = 0.

```
Solution Implementation
```

- for i in range(length -1, -1, -1): unique elements.add(nums[i]) suffix_count[i] = len(unique_elements) # Clear the set for re-use
- Java class Solution { public int[] distinctDifferenceArray(int[] nums) {

// Resultant array to store the desired distinct differences

// Traverse the nums array and calculate the distinct difference

// The current distinct difference is the size of the set

// minus the size of the suffixDistinctCount at the next index

result[i] = distinctNumbers.size() - suffixDistinctCount[i + 1];

answer[i] = len(unique_elements) - suffix_count[i + 1]

Calculate the distinct count difference for each position

```
distinctNumbers.add(nums[i]);
    suffixDistinctCount[i] = distinctNumbers.size();
// Clear the set to reuse it for the next loop
distinctNumbers.clear();
```

return result;

C++

#include <vector>

using namespace std;

#include <unordered set>

```
class Solution {
public:
    // Function to return a vector that contains the number of distinct integers
    // in the array nums after removing the elements to the right of the current element
    vector<int> distinctDifferenceArray(vector<int>& nums) {
        int n = nums.size();
        // Create a suffix array to store the count of distinct numbers from the current index to the end
        vector<int> distinctCountSuffix(n + 1);
        // Use a set to store distinct numbers to facilitate counting
        unordered_set<int> distinctNumbers;
        // Fill the suffix array with distinct number count, starting from the end of the vector
        for (int i = n - 1; i >= 0; --i) {
            distinctNumbers.insert(nums[i]);
            distinctCountSuffix[i] = distinctNumbers.size();
        // Clear the set for reuse from the start of the vector
        distinctNumbers.clear();
        // Initialize the answer vector to store the result
        vector<int> ans(n);
        // Calculate the distinct difference for each position
        for (int i = 0; i < n; ++i) {
            // Insert number into the set
            distinctNumbers.insert(nums[i]);
            // The distinct difference is the count of unique numbers we've seen so far
            // minus the count of unique numbers from the next index onwards
            ans[i] = distinctNumbers.size() - distinctCountSuffix[i + 1];
        // Return the final answer vector
        return ans;
TypeScript
function distinctDifferenceArray(nums: number[]): number[] {
    // The length of the input array
    const length = nums.length;
    // This array will hold the count of distinct numbers from the current index to the end.
    const suffixDistinctCount = new Array(length + 1).fill(0);
    // A set to keep track of distinct numbers seen so far.
    const seenNumbers = new Set<number>();
```

// Populate the suffixDistinctCount array with distinct number counts by iterating backwards.

// Iterate through the input array to calculate the difference in the distinct count

// The distinct count difference at the current index is the difference between the count of

// distinct numbers seen so far and the distinct numbers from the next index to the end.

```
# Calculate the distinct count difference for each position
for i, number in enumerate(nums):
    unique elements.add(number)
   # Calculate the difference between the number of unique elements seen
   # so far and the number of unique elements that will be seen in the suffix
    answer[i] = len(unique_elements) - suffix_count[i + 1]
```

return answer

Time Complexity

Space Complexity

which also gives O(n).

Return the answer array

Time and Space Complexity

for (let $i = length - 1; i >= 0; --i) {$

suffixDistinctCount[i] = seenNumbers.size;

// Clear the set to reuse it for the prefix pass.

// before the current index and after the current index.

// The resulting array that will hold the final difference counts.

result[i] = seenNumbers.size - suffixDistinctCount[i + 1];

def distinctDifferenceArray(self, nums: List[int]) -> List[int]:

Create an empty set to store unique elements

suffix_count[i] = len(unique_elements)

Initialize the answer array with zeros

Initialize a suffix array of length 'length + 1' with zeros

Populate the suffix count array with count of unique elements from the end

seenNumbers.add(nums[i]);

const result = new Array(length);

for (let i = 0; i < length; ++i) {

Get the length of the input list

suffix count = [0] * (length + 1)

for i in range(length -1, -1, -1):

unique elements.add(nums[i])

seenNumbers.add(nums[i]);

seenNumbers.clear();

return result;

from typing import List

length = len(nums)

unique_elements = set()

Clear the set for re-use

unique elements.clear()

answer = [0] * length

class Solution:

The first loop runs backward from n-1 to ∅. In each iteration, it adds the current element into the set s. The length of the set is then recorded in the suf array. Since adding an element to the set and checking its length are O(1) operations, this loop

The function comprises two main loops which both iterate over the array nums:

The second loop runs forward from 0 to n. Similar to the first loop, elements are added to a cleared set s, and the length difference between the sets is stored in the ans array. The complexity of this loop is also O(n).

has a complexity of O(n), where n is the number of elements in nums.

- There are no nested loops, so the overall time complexity is the sum of the two loops, which is O(n) + O(n) = O(2n). Simplified, the time complexity is O(n).
- The space complexity includes the memory used by the data structures that scale with the input size: The two sets, s and suf: s will contain at most n unique elements, thus O(n) space complexity. suf is an array of size n+1,

The distinctDifferenceArray function aims to calculate a list where each element at index i reflects the count of unique

numbers from index i to the end of the list nums, minus the count of unique numbers from index i+1 to the end.

- The ans array: Created to store the resulting differences and is of size n, contributing another O(n) space complexity. Auxiliary variables like i, x, and n: These use constant space, O(1).
- Summing these up, the total space complexity is O(n) + O(n) + O(n) which simplifies to O(3n). However, in Big O notation, constant factors are discarded, so the space complexity simplifies to O(n).