1848. Minimum Distance to the Target Element

Easy Array

Leetcode Link

Problem Description

You are provided with an array of integers called `nums`. Your task is to find an index `i` where the value at that index (`nums[i]`) is the same as the `target` value provided. Additionally, you have a starting index `start`, and you want to find the `i` that is closest to `start`. This means you want to minimize the absolute difference between `start` and `i` (`abs(i - start)`). The final output should be the minimized absolute difference. It's important to note that it is confirmed that at least one instance of the `target` value exists in the `nums` array.

Intuition

The intuition behind the solution is to iterate over each element in the array and check if it matches the `target` value. If it matches, we calculate the absolute difference between the current index `i` and the `start` index. We are interested in the minimum such difference, so we keep an ongoing record of the minimum difference calculated so far. This process involves a linear scan of the array and comparison of each element against the `target`. As we are guaranteed that `target` exists in the array, we do not need to handle cases where the target is absent. The solution approach is straightforward because the problem does not require us to optimize for time complexity beyond the linear scan, given the nature of the problem.

Solution Approach

The provided reference solution approach is a simple and direct method to solve the problem with time complexity `O(n)` and space complexity `O(1)`, where `n` is the number of elements in `nums`.

Here are the steps implemented in the solution:

1. Initialize a variable `ans` to hold the minimum distance found so far. It's initialized with `inf` (infinity), which is a placeholder for the largest possible value. This ensures that the first comparison will always replace `inf` with a valid distance.
2. Iterate through the input list `nums` using a `for` loop. The `enumerate` function is used to get both the index `i` and the value `x` at each position in the list.
3. For each element `x` and its corresponding index `i`, we check if `x` matches the `target`.
4. If a match is found, calculate the absolute difference between `i` and the given `start` index: `abs(i - start)`.
5. Update `ans` to be the minimum of the current `ans` and the newly calculated absolute difference. This step is the heart of the solution, as it maintains the smallest distance encountered as the loop progresses through the array.
6. After the loop has finished examining all elements, return the value of `ans`. At this point, `ans` contains the minimum absolute difference between the `start` index and an index `i` where `nums[i] == target`.

No additional data structures are needed, and pure iteration with basic comparisons are the only patterns used in this solution. This approach is the most optimal for this kind of problem where there isn't a pattern or structure that can be exploited to reduce the time complexity below `O(n)`.

Example Walkthrough

Let's consider a small example to illustrate the solution approach. Assume we have the following parameters:

- `nums = [4,3,2,5,3,5,1,2]`
- `target = 3`
- `start = 5`

We want to find the index `i` where `nums[i]` is equal to the target value (`3`), and which is closest to the `start` index (`5`). Following the solution steps:

1. We initialize `ans` to `inf`. At this point, `ans` would represent infinity and acts as a very high starting point for comparison.
2. As we iterate over `nums`, we will compare each element with the `target`:

Loop iteration	`i`	`x` (value at `nums[i]`)	`abs(i - start)`	`ans`
1st	0	4	5	inf
2nd	1	3	4	4 (since 4 < inf)
3rd	2	2	3	4 (since 3 > 4)
4th	3	5	2	4 (since 2 > 4)
5th	4	3	1	1 (since 1 < 4)
6th	5	5	0	1 (since 0 > 1)
7th	6	1	1	1 (since 1 >= 1)
8th	7	2	2	1 (since 2 > 1)

3. We check each time if `x == target`. When we find a match, we calculate `abs(i - start)`.
4. If a match is found:
   - For `i = 1`, `x = 3`, which matches the `target`. We calculate `abs(1 - 5) = 4` and update `ans` to `4`.
   - For `i = 4`, `x = 3` again. We calculate `abs(4 - 5) = 1` and update `ans` to `1` since `1` is less than the current `ans` of `4`.
5. We continue the process until we have iterated through the entire array. The minimum value encountered in `ans` is the one that will remain.
6. After the loop has finished, we have found that the closest index with the target value `3` relative to `start 5` is index `4` with a minimum absolute difference of `1`. Therefore, the final return value (the minimized absolute difference) is `1`.

In this example, the closest index to `start` with the target value `3` was at index `4`, which gave us the minimized absolute difference. This exemplifies the linear scan and comparison process, which results in `O(n)` time complexity and `O(1)` space complexity since no additional storage beyond a few variables is used.

Python Solution

```python
from typing import List

class Solution:
    def getMinDistance(self, nums: List[int], target: int, start: int) -> int:
        # Initialize answer with a large number (infinity)
        min_distance = float('inf')

        # Iterate through the list, enumerating it to have index and value
        for index, value in enumerate(nums):
            # Check if the current value matches the target value
            if value == target:
                # Update min_distance with the smaller value between the current min_distance
                # and the absolute difference between current index and start index
                min_distance = min(min_distance, abs(index - start))

        # Return the minimum distance found
        return min_distance

# Example usage:
# sol = Solution()
# result = sol.getMinDistance([1, 2, 3, 4, 5], 5, 3)
# print(result)  # Output will be 1, since the distance between index 3 and the closest 5 is 1.
```

Java Solution

```java
class Solution {
    public int getMinDistance(int[] nums, int target, int start) {
        // Get the length of the input array
        int arrayLength = nums.length;
        // Initialize the answer with the maximum possible value
        int minimumDistance = arrayLength;

        // Iterate through the array to find the elements equal to the target
        for (int i = 0; i < arrayLength; ++i) {
            // Check if the current element equals the target
            if (nums[i] == target) {
                // Update the minimum distance if the current distance is smaller than the previously computed one
                minimumDistance = Math.min(minimumDistance, Math.abs(i - start));
            }
        }
        // Return the smallest distance found
        return minimumDistance;
    }
}
```

C++ Solution

```cpp
#include <vector>    // Required for using the vector container
#include <algorithm>    // Required for 'min' function
#include <cmath>    // Required for 'abs' function

class Solution {
public:
    // Function to find the minimum distance to the target from the start index
    int getMinDistance(vector<int>& nums, int target, int start) {
        int size = nums.size();  // Get the size of the input vector 'nums'
        int minDistance = size;  // Initialize minimum distance with the maximum possible value (size of the vector)

        // Loop through all elements in the nums vector
        for (int i = 0; i < size; ++i) {
            // Check if the current element is equal to the target
            if (nums[i] == target) {
                // Update the minimum distance found so far
                minDistance = min(minDistance, abs(i - start));
            }
        }

        // Return the minimum distance to the target from the start index
        return minDistance;
    }
};
```

Typescript Solution

```typescript
import * as util from "util";  // TypeScript doesn't natively import min and abs, so we would typically use a utility library or img

// Function to find the minimum distance to the target from the start index
function getMinDistance(nums: number[], target: number, start: number): number {
  const size: number = nums.length;  // Get the size of the input array 'nums'
  let minDistance: number = size;  // Initialize minimum distance with the maximum possible value (size of the array)

  // Loop through all elements in the nums array
  for (let i = 0; i < size; ++i) {
    // Check if the current element is equal to the target
    if (nums[i] === target) {
      // Update the minimum distance found so far using Math.min and Math.abs for minimum and absolute value respectively
      minDistance = Math.min(minDistance, Math.abs(i - start));
    }
  }

  // Return the minimum distance to the target from the start index
  return minDistance;
}

// Example usage:
// Uncomment the line below to test the function with an example input
// console.log(getMinDistance([1, 2, 3, 4], 3, 2));
```

Time and Space Complexity

The time complexity of the given code is `O(n)`, where `n` is the length of the `nums` list. This is because the code iterates through each element of `nums` once to check if it is equal to `target` and, if so, calculates the distance from the `start` index. The `min` function, called for each element of the list, operates in constant time `O(1)`; hence, it does not affect the overall linear complexity.

The space complexity of the code is `O(1)`. This is because the space used does not grow with the size of the input list. The `ans` variable takes constant space, and there are no additional data structures used that scale with the input size.