My Experience with LeetCode

Introduction

LeetCode is a platform for preparing technical coding interviews. It offers a vast collection of coding problems across various difficulty levels, ranging from easy to hard. My experience with LeetCode has been enriching and instrumental in sharpening my problem-solving skills. In this report, I will summarize three problems I solved on LeetCode, including the approaches I used and their solutions.

Problem 1: Two Sum

Problem Description

Given an array of integers nums and an integer target, return indices of two numbers such that they add up to target. Assume that each input would have exactly one solution, and you may not use the same element twice.

Example:

plaintext

Input: nums = [2, 7, 11, 15], target = 9

Output: [0, 1]

Explanation: Because nums[0] + nums[1] == 9, we return [0, 1].

Approach

To solve the Two Sum problem, I used a hash table (dictionary) to store the difference between the target and the current element as the key, and the index of the current element as the value. This allows for checking in constant time if the complement of the current element exists in the hashtable.

Solution

```
def twoSum(nums, target):
    num_map = {} To store the complement and its index
    for i, num in enumerate(nums):
        Complement = target - num
        if complement in num_map:
            return [num_map[complement], i]
        num_map[num] = i
```

Explanation

1. Initialize an empty dictionary num_map.
2. Iterate through the list nums with index i and value num.
3. Calculate the complement as target - num.
4. Check if the complement exists in num_map. If it does, return the indices.
5. If not, store num and its index i in num_map.

Problem 2: Longest Substring Without Repeating Characters

Probem Description

Given a string s, find the length of the longest substring without repeating characters.

Example:

plaintext

Input: s = "abcabcbb"

Output: 3

Explanation: The answer is "abc", with the length of 3.

Approach

I used the sliding window technique along with a set to keep track of characters in the current window. By adjusting the window size dynamically, I ensured that all characters in the window were unique.

Solution

```
def lengthOfLongestSubstring(s):
    char_set = set()
    left = 0
    max_length = 0
    for right in range(len(s)):
        while s[right] in char_set:
            char_set.remove(s[left])
            left += 1
        char_set.add(s[right])
        max_length = max(max_length, right - left + 1)
    return max_length
```

Explanation

1. Initialize an empty set char_set to store unique characters.
2. Use two pointers, left and right, to represent the current window.
3. Iterate with the right pointer through the string s.
4. If s[right] is in char_set, remove characters from char_set and move the left pointer until s[right] is unique.
5. Add s[right] to char_set.
6. Update max_length with the maximum length of the current window.

Problem 3: Merge Two Sorted Lists

Problem Description:

You are given the heads of two sorted linked lists list1 and list2. Merge the two lists in a one sorted list. The list should be made by splicing together the nodes of the first two lists.

Example:
Input: list1 = [1,2,4], list2 = [1,3,4]
output: [1,1,2,3,4,4]

Approach

To merge two sorted linked lists, I used a dummy node to simplify the process of meoging. By iterating through both lists and linking nodes in sorted order, I ensured the merged list remained sorted.

Solution

```
Class ListNode
  def_int_ (self, val=0, next=None):
      Self.val = val
      Self.next= next
def mergeTwoLists(list1, list2):
    dummy = ListNode()
    Current = dummy

    while list1 and list2:
      if list1.val < list2.val:
        current.next = list1
        list1 = list1.next
      else:
        current.next = list2
        list2 = list2.next
      Current = current.next

    if list1:
      current.next = list1

    elif list2:
      current.next = list2

    return dummy.next
```

Explanation:

1. Create a dummy node to serve as the starting point of the merged list.
2. Use a pointer current to track the end of the merged list.
3. Iterate through list1 and list2, Comparing the values of their nodes.
4. Link the node with the smaller value to current and move the pointer of that list forward.
5. Once one list is exhausted, link the remaining nodes of the other list to current.
6. Return the next node of the dummy, which is the head of the merged list.