
Intuition

The problem requires identifying the first character in the string that appears twice. The goal is to efficiently track character occurrences and find the second appearance of a character as soon as possible. By using a dictionary to count occurrences, we can keep track of each character we see while scanning through the string in a single pass. As soon as a character's count reaches two, we know it's the first character to appear twice, allowing us to return it immediately without further processing.

Approach

1. **Initialize an Empty Dictionary**: Create an empty dictionary called store to track the count of each character in the string.
2. **Loop Through Characters**: Iterate through each character char in the string s.

- For each character, update its count in the dictionary. If it's not already in the dictionary, initialize it to 0 and then add 1.

3. **Check Count for Repetition**: After updating the count, check if it has reached 2.

- If it has, this means char is the first character to appear twice, so return it immediately.

4. **End of Loop**: No further action is needed since the problem guarantees there will always be a character that appears twice.

Complexity Analysis

- **Time Complexity:** The time complexity of this solution is O(n), where n is the length of the string s. This is because we only make a single pass through the string, examining each character once. For each character, we perform a constant-time operation to update its count in the dictionary and check if it appears twice. Therefore, the time complexity remains linear in relation to the length of the input string.
- **Space Complexity:** The space complexity is O(k), where k represents the number of unique characters in the string s. In the worst case, each character in the string is unique up until the first repeated character, meaning the dictionary will store at

most k entries. Since k is always less than or equal to n, the space complexity is linear in the number of unique characters in the input.

This approach is optimal for the problem constraints and performs well with minimal space usage relative to the input size.