2124. Check if All A's Appears Before All B's

Easy String

Leetcode Link

Problem Description

In this problem, we are given a string `s` that contains only two characters: 'a' and 'b'. The objective is to determine whether all the 'a' characters in the string appear before any 'b' character. If this condition is met, the function should return `true`, indicating that the string maintains the order where all 'a's come before all 'b's. If this condition is not met, the function should return `false`. This is a straightforward problem that checks for a specific ordering of characters within a string.

Intuition

The intuition behind the solution is based on a simple observation about the string's character order. If at any point in the string a 'b' appears before an 'a', then not every 'a' is before every 'b', and thus we should return `false`. A simple way to check this is to look for the substring "ba" within `s`. If "ba" is found, it means there is an occurrence where 'b' comes directly before 'a', which violates our condition, and as a result, we should return `false`. On the other hand, if "ba" is not found, it means that all 'a' characters, if present, come before any 'b' character, enabling us to return `true`.

The provided solution takes advantage of the built-in string operation in Python that checks for the presence of a substring within a string. By using the expression `"ba" not in s`, we verify if "ba", which is the pattern we don't want to see, is absent from `s`. If the pattern is absent, it means that the string `s` satisfies the condition and therefore the function returns `true`. If the pattern is present, the expression evaluates to `false`, which means the condition is not met, and the function will return `false`.

Solution Approach

The implementation of the solution uses a straightforward approach without the need for complex algorithms or data structures. It leverages a simple string scanning feature intrinsic to Python.

Algorithm:

1. We scan the string `s` for the substring "ba".
2. If "ba" is found, it indicates that there is at least one 'b' that comes before an 'a'.
3. The implementation checks for the non-existence of "ba" to ensure all 'a's come before any 'b's.

Patterns Used:

- **String Search Pattern**: This solution uses a string search to find a specific sequence of characters within another string. In Python, this is commonly done using the `in` keyword, which checks for the membership of a substring within a larger string.

Code Explanation:

- `return "ba" not in s`: This line of code is the entire implementation. It checks whether the substring "ba" does not exist in the input string `s`. If the substring "ba" is not found, it implies that the string `s` maintains the correct order of characters as per the problem statement and thus returns `true`. Otherwise, it returns `false`.

This method is extremely efficient because it utilizes Python's built-in string operations, which are highly optimized for these kinds of operations. No additional space is required, as we are not storing any additional data structures, making the space complexity O(1). The time complexity is O(n), where n is the length of the string `s`, since the search operation has to potentially check each character in the string one by one in the worst-case scenario.

Example Walkthrough

Let's consider the example string `s = "aaabb"`. To illustrate the solution approach, we will apply the algorithm step by step.

1. The string `s` is scanned for the substring "ba".
2. As we go through the string from the start, we see that it begins with 3 'a' characters followed by 2 'b' characters. The substring "ba" does not occur at any point in this string. There is no 'b' that appears before an 'a'. Thus, the condition for the string to maintain that all 'a's come before any 'b's is satisfied.
3. Since the substring "ba" is not found in `s`, we conclude that all 'a' characters appear before all 'b' characters.
4. The expression `"ba" not in s` is evaluated. Since "ba" indeed does not exist in `s`, the expression evaluates to `true`.
5. Since the expression evaluates to `true`, the function would return `true`, correctly indicating that the input string `s` maintains the order where all 'a's come before any 'b's.

In this example, we have followed the string search pattern methodology as detailed in the solution approach. The immediate absence of "ba" signifies that the order is maintained, thereby allowing the approach to quickly determine the correct result without the need for additional checks or data structures.

Python Solution

```python
class Solution:
    def check_string(self, s: str) -> bool:
        # Check if the string 's' does not contain the substring "ba"
        # If "ba" is not present, it means all 'b's are after 'a's which is valid
        return "ba" not in s

```

Java Solution

```java
// Class definition
class Solution {
    // Method to check if the input string 's' does NOT contain the substring "ba"
    public boolean checkString(String s) {
        // If the string 's' contains "ba", return false
        // otherwise, return true
        return !s.contains("ba");
    }
}

```

C++ Solution

```cpp
class Solution {
public:
    // Function to check if the string 's' contains the substring "ba".
    // It returns true if "ba" is not present; otherwise, it returns false.
    bool checkString(string s) {
        // Find the first occurrence of "ba" in the string 's'
        size_t position = s.find("ba");

        // If "ba" is not found, string::npos is returned
        // string::npos is a constant representing a non-position
        // If "ba" is not present, we return true;
        // otherwise, we return false
        return position == string::npos;
    }
};

```

Typescript Solution

```typescript
// Function to check if the string 's' contains the substring "ba".
// It returns true if "ba" is not present; otherwise, it returns false.
function checkString(s: string): boolean {
    // Find the first occurrence of "ba" in the string 's'
    let position: number = s.indexOf("ba");

    // If "ba" is not found, indexOf returns -1
    // If "ba" is not present, we return true;
    // otherwise, we return false
    return position === -1;
}

```

Time and Space Complexity

The time complexity of the code is `O(n)`, where `n` is the length of the string `s`. This is because in the worst case, the `in` operator needs to check each pair of consecutive characters in the string until it either finds the substring "ba" or confirms that the substring is not present.

The space complexity of the code is `O(1)` since the amount of additional memory used does not depend on the size of the input string. The method uses a fixed amount of space to store the return value and any temporary variables, which does not scale with the input size.