551. Student Attendance Record I

Easy String

Leetcode Link

Problem Description

In this problem, we receive a string `s` representing a student's attendance record. Each character in this record signifies whether the student was Absent (`'A'`), Late (`'L'`), or Present (`'P'`) on a given day. The goal is to determine whether the student qualifies for an attendance award based on this record. The criteria for qualification are simple:

- First, the student must have been absent (`'A'`) for strictly fewer than 2 days in total.
- Second, the student must never have been late (`'L'`) for 3 or more consecutive days.

The function must return `true` if the student meets both criteria for the attendance award, otherwise, it should return `false`.

Intuition

The intuition behind the provided solution is straightforward. Since the criteria are not compound and don't require advanced checks, we can use two simple checks directly on the string:

1. Counting the number of 'A's: We ensure that the count of absences is less than 2, which means either 0 or 1 absence is acceptable.
2. Checking for 'LLL': To verify the student was never late for 3 or more consecutive days, we look for the substring `'LLL'` in `s`. If it's not found, the student has never been late for 3 consecutive days.

The solution leverages the inbuilt `count` method to count the absences and the `in` operator to check for the presence of three consecutive 'L's. Since the problem statement requires both conditions to be true, we use the logical AND operator (`and`) to combine the two conditions. If both conditions evaluate to `true`, the overall expression returns `true`, indicating the student is eligible for the award.

Solution Approach

The implementation of the solution is straightforward and utilizes two built-in Python features, which are simple to understand and do not require additional data structures or complex algorithms:

1. **`.count()` Method:** This is a built-in string method in Python that counts the number of times a substring appears in the string. In our solution, `s.count('A')` is used to count the number of 'A's, which represent absences in the attendance record. We check that this count is less than 2.

2. **Substring Checking With `in` Operator**: We use the `in` keyword to check if a substring appears within a larger string. In this case, we're checking if `'LLL'` (representing three consecutive late days) is a substring of `s`. If `'LLL'` is not found, it means the student has not been late for 3 or more consecutive days.

By combining these two checks with the logical AND operator (`and`), the solution checks both conditions stipulated in the problem description. Specifically, the code snippet `s.count('A') < 2 and 'LLL' not in s` will evaluate to `true` only if both conditions are met:

- There are fewer than 2 absences, which means the `s.count('A') < 2` condition is `true`.
- There are no instances of 3 consecutive lates, which means the `'LLL' not in s` condition is `true`.

The function `checkRecord` hence returns a single boolean value, which directly answers whether the student is eligible for the attendance award according to the given rules. The efficiency of this approach lies in the fact that both checks are performed in constant time—there's no loop or iteration across the string, making the time complexity O(n) where n is the length of the string, with the space complexity being O(1) as no extra space is used.

Example Walkthrough

Let's consider the string `s = "PPALLP"` as a student's attendance record for illustration purposes.

1. **Counting Absences:** We apply `s.count('A')` to count how many times `'A'` appears in `s`. For our example, `s.count('A')` will return `1` since there is only one `'A'` character in the string `PPALLP`.

2. **Checking for Three Consecutive 'L's:** We then check if the substring `'LLL'` exists in `s` by using the `in` operator: `'LLL' not in s`. For our example, `'LLL'` does not appear in `PPALLP`. Therefore, `'LLL' not in s` would be `true`.

Now we apply the logical AND operator to combine these two checks:

- Since `s.count('A') < 2` evaluates to `true` (1 < 2 is true),
- and `'LLL' not in s` evaluates to `true`,

the combined condition is `true` AND `true`, which yields `true`. Therefore, the function `checkRecord("PPALLP")` would return `true`, indicating that the student qualifies for an attendance award based on the given record.

Python Solution

```python
class Solution:
    def checkRecord(self, s: str) -> bool:
        """
        Check if a student's attendance record is eligible for an award.
        The record is eligible if it contains:
        - Less than two 'A' (Absent) records.
        - No more than two continuous 'L' (Late) records.

        :param s: A string representing the student's attendance record.
        :type s: str
        :return: True if the record is eligible, False otherwise.
        :rtype: bool
        """

        # Check for less than two Absents
        less_than_two_absents = s.count('A') < 2

        # Check for not having three continuous Lates
        no_three_continuous_lates = 'LLL' not in s

        # Return True if both conditions are met, otherwise False
        return less_than_two_absents and no_three_continuous_lates

# Example usage:
# creating an instance of Solution class
solution = Solution()
# invoking the checkRecord method with a sample record string
eligible = solution.checkRecord("PPALLP")  # This should return True

```

Java Solution

```java
class Solution {
    // This method checks if a student's attendance record is eligible for an award.
    // The record is eligible if it contains at most one 'A' (absence)
    // and it does not contain a 'LLL' (late three times in a row).
    public boolean checkRecord(String attendanceRecord) {
        // Check that there is at most one 'A' in the string by confirming that
        // the first index of 'A' is the same as the last index of 'A'.
        boolean hasAtMostOneAbsence = attendanceRecord.indexOf("A") == attendanceRecord.lastIndexOf("A");

        // Check that 'LLL' does not appear in the string.
        boolean noThreeConsecutiveLates = !attendanceRecord.contains("LLL");

        // The student is eligible for an award if both conditions are true.
        return hasAtMostOneAbsence && noThreeConsecutiveLates;
    }
}

```

C++ Solution

```cpp
class Solution {
public:
    // This method checks if the student's record is eligible for an award.
    // A record is eligible if it contains fewer than two 'A's (absences)
    // and does not contain three consecutive 'L's (lates).
    bool checkRecord(string record) {
        // Check for the number of absences
        // The record is ineligible if 'A's appear 2 or more times
        bool isAbsenceLessThanTwo = count(record.begin(), record.end(), 'A') < 2;

        // Check for consecutive lates
        // The record is ineligible if 'LLL' is found
        bool hasNoConsecutiveLates = record.find("LLL") == string::npos;

        // The record is eligible only if both conditions are satisfied
        return isAbsenceLessThanTwo && hasNoConsecutiveLates;
    }
};

```

Typescript Solution

```typescript
// Function to determine if a student's attendance record is eligible
// The record is considered eligible if it contains at most one 'A' (absence)
// and does not contain three consecutive 'L' (late).
function checkRecord(attendanceRecord: string): boolean {
    // Check if the record contains at most one 'A' by comparing the first and last occurrence.
    // If they are the same, there is either one or zero 'A' in the record.
    const isSingleAbsenceOrLess = attendanceRecord.indexOf('A') === attendanceRecord.lastIndexOf('A');

    // Check if the record does not contain 'LLL'.
    const doesNotContainThreeLates = attendanceRecord.indexOf('LLL') === -1;

    // Return true if both conditions are met, otherwise return false.
    return isSingleAbsenceOrLess && doesNotContainThreeLates;
}

```

Time and Space Complexity

The time complexity of the provided function is `O(n)`, where `n` is the length of the string `s`. This is because both `count` and `in` methods must potentially examine each character in the string – `count` to tally occurrences of 'A' and the `in` operation to check for the substring 'LLL'.

The space complexity of the function is `O(1)` because no additional space is used that is dependent on the input size; the storage used is constant, related only to the single boolean result and fixed string checks, and does not grow with the size of `s`.