551. Student Attendance Record I

April 15, 2017 | 7.7K views

Average Rating: 4.55 (11 votes)

You are given a string representing an attendance record for a student. The record only contains the following three characters:

1. **'A'** : Absent.
2. **'L'** : Late.
3. **'P'** : Present.

A student could be rewarded if his attendance record doesn't contain **more than one 'A' (absent)** or **more than two continuous 'L' (late)**.

You need to return whether the student could be rewarded according to his attendance record.

Example 1:

```
Input: "PPALLP"
Output: True
```

Example 2:

```
Input: "PPALLL"
Output: False
```

Solution

Approach #1 Simple Solution [Accepted]

One simple way of solving this problem is to count number of $A's$ in the string and check whether the string LLL is a substring of a given string. If number of $A's$ is less than 2 and LLL is not a subtring of a given string then return $true$, otherwise return $false$.

$indexOf$ method can be used to check substring in a string. It return the index within this string of the first occurrence of the specified character or -1, if the character does not occur.

Java

Copy

```
public class Solution {
    public boolean checkRecord(String s) {
        int count=0;
        for(int i=0;i<s.length();i++)
            if(s.charAt(i)=='A')
                count++;
        return count<2 && s.indexOf("LLL")<0;
    }
}
```

Complexity Analysis

- Time complexity : $O(n)$. Single loop and indexOf method takes $O(n)$ time.
- Space complexity : $O(1)$. Constant space is used.

Approach #2 Better Solution [Accepted]

Algorithm

One optimization of above method is to break the loop when count of $A's$ becomes 2.

Java

Copy

```
public class Solution {
    public boolean checkRecord(String s) {
        int count=0;
        for(int i=0;i<s.length() && count<2 ;i++)
            if(s.charAt(i)=='A')
                count++;
        return count<2 && s.indexOf("LLL")<0;
    }
}
```

Complexity Analysis

- Time complexity : $O(n)$. Single loop and indexOf method takes $O(n)$ time.
- Space complexity : $O(1)$. Constant space is used.

Approach #3 Single pass Solution (Without indexOf method) [Accepted]

Algorithm

We can solve this problem in a single pass without using indexOf method. In a single loop we can count number of $A's$ and also check the substring LLL in a given string.

Java

Copy

```
public class Solution {
    public boolean checkRecord(String s) {
        int countA = 0;
        for (int i = 0; i < s.length() && countA < 2; i++) {
            if (s.charAt(i) == 'A')
                countA++;
            if (i <= s.length() - 3 && s.charAt(i) == 'L' && s.charAt(i + 1) == 'L' && s.charAt(i + 2) ==
'L')
                return false;
        }
        return countA < 2;
    }
}
```

Complexity Analysis

- Time complexity : $O(n)$. Single loop upto string length is used.
- Space complexity : $O(1)$. Constant space is used.

Approach #4 Using Regex [Accepted]

Algorithm

One interesting solution is to use regex to match the string. Java provides the java.util.regex package for pattern matching with regular expressions. A regular expression is a special sequence of characters that helps you match or find other strings or sets of strings, using a specialized syntax held in a pattern.

Following are the regex's used in this solution:

```
. : Matches any single character except newline.

* : Matches 0 or more occurrences of the preceding expression.

.* : Matches any string

a|b : Matches either a or b
```

$matches$ method is used to check whether or not the string matches the given regular expression.

Regular Expression of the string containing two or more than two $A's$ will be $.*A.*A.*$ and the regular expression of the string containing substring LLL will be $.*LLL.*$. We can merge this two regex using | and form a regex of string containing either more than one A or containing substring LLL. Then regex will look like: $.*(A.*A|LLL).*$. We will return true only when the string doesn't matches this regex.

Java

Copy

```
public class Solution {
    public boolean checkRecord(String s) {
        return !s.matches(".*(A.*A|LLL).*");
    }
}
```

Complexity Analysis

- Time complexity : $O(n)$. $matches$ method takes $O(n)$ time.
- Space complexity : $O(1)$. No Extra Space is used.

Rate this article:

Comments: 4

Sort By

Type comment here... (Markdown is supported)

Preview Post

calvinchankf ★ 2997 June 4, 2019 7:46 AM

The regex is impressive but I dont think i can remember the regex syntax during an interview

18 Share Reply

Username1604 ★ 45 June 14, 2020 11:15 PM

```
class Solution:
    def checkRecord(self, s: str) -> bool:
        return s.count('A')<=1 and not 'LLL' in s
```

1 Share Reply

MaU ★ 85 October 31, 2019 7:42 PM

I would use a boolean value to store whether 'A' has come up yet or not, and return false accordingly, rather than using an integer to count the occurences of 'A' and adding a condition to be checked in every iteration of the 'for' loop. It is more efficient this way.

In C++:

Read More

1 Share Reply

prithivm ★ 4 March 16, 2020 11:28 PM

```
class Solution:
    def checkRecord(self, s: str) -> bool:
        out = True
        if s.count('A') > 1:
```

Read More

0 Share Reply