36 学生出勤记录

更新时间：2019-09-26 09:49:27

> 横眉冷对千夫指，俯首甘为孺子牛。
>
> ——鲁迅

刷题内容

> **难度: Hard**
>
> 题目链接：https://leetcode-cn.com/problems/student-attendance-record-ii/

题目描述

给定一个正整数 n，返回长度为 n 的所有可被视为可奖励的出勤记录的数量。 答案可能非常大，你只需返回结果mod 109 + 7的值。

学生出勤记录是只包含以下三个字符的字符串：

'A' : Absent，缺勤
'L' : Late，迟到
'P' : Present，到场
如果记录不包含多于一个'A'（缺勤）或超过两个连续的'L'（迟到），则该记录被视为可奖励的。

示例 1:

输入: n = 2
输出: 8
解释：
有8个长度为2的记录将被视为可奖励：
"PP" , "AP", "PA", "LP", "PL", "AL", "LA", "LL"
只有"AA"不会被视为可奖励，因为缺勤次数超过一次。
注意：n 的值不会超过100000。

解题方案

思路1 时间复杂度: O(3^N) 空间复杂度: O(3^N)

我们首先想到可以穷举所有的可能性，这里采用dfs的方法

因为我们最终的字符串长度为n，且每一位上都有3种可能性，所以我们的时间复杂度为O(3^N)，这样很明显会超时

Python

超时

```python
class Solution:
    def checkRecord(self, n: int) -> int:
        M = 10 ** 9 + 7
        self.res = 0
        def dfs(cur, cnt_a):
            if len(cur) == n: # 如果出勤记录长度达到了n就退出
                self.res += 1
                self.res %= M
                return
            if cnt_a < 1: # 如果'A'的个数少于1，我们可以允许有一次'A'
                dfs(cur+'A', cnt_a+1)
            if len(cur) < 2 or not (cur[-1] == cur[-2] == 'L'): # 如果没有连续两次的'L'，我们可以加上一次'L'
                dfs(cur+'L', cnt_a)
            dfs(cur+'P', cnt_a) # 任何时候我们加上一个'P'都是可行的
        dfs('', 0)
        return self.res % M
```

Go

超时

```go
import "math"
var res = 0
var M = int(math.Pow10(9)) + 7

func dfs(cur string, cntA int, n int) {
    if len(cur) == n { // 如果出勤记录长度达到了n就退出
        res += 1
        res %= M
        return
    }
    if cntA < 1 { // 如果'A'的个数少于1，我们可以允许有一次'A'
        dfs(cur+"A", cntA + 1, n)
    }
    // 如果没有连续两次的'L'，我们可以加上一次'L'
    if len(cur) < 2 || !(cur[len(cur)-1] == cur[len(cur)-2] && cur[len(cur)-1] == 'L') {
        dfs(cur+"L", cntA, n)
    }
    dfs(cur+"P", cntA, n) // 任何时候我们加上一个'P'都是可行的
}

func checkRecord(n int) int {
    dfs("", 0, n)
    tmp := res
    res = 0
    return tmp % M
}
```

Java

超时

```
class Solution {
    int res = 0;
    int M = (int) Math.pow(10, 9) + 7;
    public void dfs(String cur, int cntA, int n) {
        if (cur.length() == n) { // 如果出勤记录长度达到了n就退出
            res += 1;
            res %= M;
            return;
        }
        if (cntA < 1) { // 如果'A'的个数少于1，我们可以允许有一次'A'
            dfs(cur+"A", cntA + 1, n);
        }
        // 如果没有连续两次的'L'，我们可以加上一次'L'
        if ((cur.length() < 2) || !(cur.charAt(cur.length()-1) == cur.charAt(cur.length()-2) && cur.charAt(cur.length()-1) == 'L')) {
            dfs(cur+"L", cntA, n);
        }
        dfs(cur+"P", cntA, n); // 任何时候我们加上一个'P'都是可行的
    }
    public int checkRecord(int n) {
            dfs("", 0, n);
            return res % M;
    }
}
```

C++

超时

```
class Solution {
public:
    int res = 0;
    int M = (int) 1E9 + 7;
    void dfs(string cur, int cntA, int n) {
        if (cur.length() == n) { // 如果出勤记录长度达到了n就退出
            res += 1;
            res %= M;
            return;
        }
        if (cntA < 1) { // 如果'A'的个数少于1，我们可以允许有一次'A'
            dfs(cur+"A", cntA + 1, n);
        }
        // 如果没有连续两次的'L'，我们可以加上一次'L'
        if ((cur.length() < 2) || !(cur[cur.length()-1] == cur[cur.length()-2] && cur[cur.length()-1] == 'L')) {
            dfs(cur+"L", cntA, n);
        }
        dfs(cur+"P", cntA, n); // 任何时候我们加上一个'P'都是可行的
    }
    int checkRecord(int n) {
        dfs("", 0, n);
        return res % M;
    }
};
```

思路2 时间复杂度: O(N) 空间复杂度: O(N)

我们想象一下， 假设我们已经拿到了一个可奖励的的长度为i的记录，我们需要再末尾再加上一个字符来使得新的记录还是可奖励的，怎么做呢？

很显然，跟思路一样，我们可以选择在末尾加上一个'L'， 'P'或者一个'A'，但是我们怎么知道前面是否已经有'A'了呢？

何不如我们先假设我们已经得到了一个可奖励的长度为i的，且不包含‘A’的记录呢？
假设这样的记录有dp[i]种，那么dp[i]可以怎么算出来呢？

因为dp[i]里面是没有’A’的，所以dp[i]只能是以’L’或者’P’结尾，因此：

dp[i] = dp[i].endwith(‘P’) + dp[i].endwith(‘L’)
= dp[i].endwith(‘P’) + dp[i].endwith(‘PL’) + dp[i].endwith(‘LL’)
= dp[i].endwith(‘P’) + dp[i].endwith(‘PL’) + dp[i].endwith(‘PLL’) + dp[i].endwith(‘LLL’)

又因为dp[i].endwith(‘LLL’)是不可以被奖励的，因为有3个连续的’L’了

所以dp[i] = dp[i].endwith(‘P’) + dp[i].endwith(‘PL’) + dp[i].endwith(‘PLL’)

其中：

- dp[i].endwith(‘P’)的值其实就是dp[i-1]
- dp[i].endwith(‘PL’)的值其实就是dp[i-2]
- dp[i].endwith(‘PLL’)的值其实就是dp[i-3]

因此我们得到：dp[i] = dp[i-1] + dp[i-2] + dp[i-3]

所以我们可以把dp[1]到dp[n]全部都算出来，然后呢？

因为dp[i]中是不包含’A’的，所以我们可以把’A’放在记录的index 0 到n-1之间任意一个位置，最终就可以算出我们总的可以奖励的长度为n的记录个数啦

Python

beats 100%

```python
class Solution:
    def checkRecord(self, n: int) -> int:
        dp = [1, 2, 4] # 这3个数字是根据规律推出来的, dp[0] = 1, dp[1] = 2, dp[2] = 4
        M = 10 ** 9 + 7
        for i in range(2, n):
            dp.append((dp[i] + dp[i-1] + dp[i-2]) % M)
        res = 0
        for i in range(n+1):
            if i == n:
                res += dp[n]
            else:
                res += (dp[i] * dp[n-1-i]) % M # i 代表 'A' 前面的长度，n-1-i 代表 'A' 后面的长度
            res %= M
        return res
```

Go

beats 100%

```
func checkRecord(n int) int {
    dp := []int{1, 2, 4} // 这3个数字是根据规律推出来的, dp[0] = 1, dp[1] = 2, dp[2] = 4
    M := int(math.Pow10(9)) + 7
    for i := 2; i < n; i++ {
        dp = append(dp, (dp[i] + dp[i-1] + dp[i-2]) % M)
    }
    res := 0
    for i := 0; i < n + 1; i++ {
        if i == n {
            res += dp[n]
        } else {
            res += (dp[i] * dp[n-1-i]) % M // i 代表 'A' 前面的长度，n-1-i 代表 'A' 后面的长度
        }
        res %= M
    }
    return res
}
```

Java

beats 100%

```
class Solution {
    public int checkRecord(int n) {
        int M = (int) Math.pow(10, 9) + 7;
        List<Long> dp = new ArrayList<Long>();
        dp.add((long) 1); // 这3个数字是根据规律推出来的, dp[0] = 1, dp[1] = 2, dp[2] = 4
        dp.add((long) 2);
        dp.add((long) 4);
        for (int i = 2; i < n; i++) {
            dp.add((dp.get(i) + dp.get(i-1) + dp.get(i-2)) % M);
        }
        int res = 0;
        for (int i = 0; i < n + 1; i++) {
            if (i == n) {
                res += dp.get(n);
            } else {
                res += (dp.get(i) * dp.get(n-1-i)) % M; // i 代表 'A' 前面的长度，n-1-i 代表 'A' 后面的长度
            }
            res %= M;
        }
        return res;
    }
}
```

C++

beats 100%

```
class Solution {
public:
    int checkRecord(int n) {
        int M = (int) 1e9 + 7;
        vector<long> dp; // 这3个数字是根据规律推出来的, dp[0] = 1, dp[1] = 2, dp[2] = 4
        dp.push_back(1);
        dp.push_back(2);
        dp.push_back(4);
        for (int i = 2; i < n; i++) {
            dp.push_back((dp[i]+ dp[i-1] + dp[i-2]) % M);
        }
        int res = 0;
        for (int i = 0; i < n + 1; i++) {
            if (i == n) {
                res += dp[n];
            } else {
                res += (dp[i] * dp[n-1-i]) % M; // i 代表 'A' 前面的长度，n-1-i 代表 'A' 后面的长度
            }
            res %= M;
        }
        return res;
    }
};
```

总结

- 穷举法有时候可以帮助我们看出规律，有些题目你就按照题目要求列出个几个数字来，你就能观察出规律，这也被称为打表法

}

35 地下城游戏

37 和最少为K的最短子数组