# 题目

可以用字符串表示一个学生的出勤记录，其中的每个字符用来标记当天的出勤情况（缺勤、迟到、到场）。记录中只含下面三种字符：

'A'：Absent，缺勤

'L'：Late，迟到

'P'：Present，到场

如果学生能够 同时 满足下面两个条件，则可以获得出勤奖励：

按 总出勤 计，学生缺勤（'A'）严格 少于两天。

学生 不会 存在 连续 3 天或 连续 3 天以上的迟到（'L'）记录。

给你一个整数 n ，表示出勤记录的长度（次数）。请你返回记录长度为 n 时，可能获得出勤奖励的记录情况 数量 。答案可能很大，所以返回对 109 + 7 取余 的结果。

示例 1：

输入：n = 2

输出：8

解释：

有 8 种长度为 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，因为缺勤次数为 2 次（需要少于 2 次）。

示例 2：

输入：n = 1

输出：3

示例 3：

输入：n = 10101

输出：183236316

提示：

1 <= n <= 105

# 分析

## 方法一：动态规划

要解决“计算长度为n的有效出勤记录数量”问题，核心思路是动态规划（DP），通过定义状态跟踪缺勤次数和连续迟到次数，逐步推导出所有符合条件的记录数量。

解题思路

1、状态定义：

有效出勤记录需满足两个条件：缺勤（'A'）次数≤1，连续迟到（'L'）次数≤2。因此，定义三维DP状态：

- dp[i][a][l]：表示长度为i的记录中，包含a次缺勤（a=0或1）和连续l次迟到（l=0、1或2）的有效记录数量。

2、状态转移：

对于第i+1天的记录，可根据当天的出勤情况（'P'、'A'、'L'）更新状态：

- 当天为'P'：缺勤次数不变，连续迟到次数重置为0。

dp[i+1][a][0] += dp[i][a][l]（a=0,1；l=0,1,2）。

- 当天为'A'：仅当当前缺勤次数a=0时可添加，缺勤次数变为1，连续迟到次数重置为0。

dp[i+1][1][0] += dp[i][0][l]（l=0,1,2）。

- 当天为'L'：连续迟到次数加1（但不超过2），缺勤次数不变。

dp[i+1][a][l+1] += dp[i][a][l]（a=0,1；l=0,1）。

3、初始条件：

- 长度为1的记录：

dp[1][0][0] = 1（'P'），dp[1][0][1] = 1（'L'），dp[1][1][0] = 1（'A'）。

4、结果计算：

长度为n的所有有效记录数量为所有dp[n][a][l]的总和（a=0,1；l=0,1,2）。

代码：

class Solution {

public:

const int MOD = 1e9 + 7;

int checkRecord(int n) {

if (n == 0) return 0;

// 优化空间：仅用前一天的状态（prev\_dp）计算当前天（curr\_dp）

vector<vector<int>> prev\_dp(2, vector<int>(3, 0));

// 初始条件：n=1时的状态

prev\_dp[0][0] = 1; // 'P'：0次缺勤，0次连续迟到

prev\_dp[0][1] = 1; // 'L'：0次缺勤，1次连续迟到

prev\_dp[1][0] = 1; // 'A'：1次缺勤，0次连续迟到

for (int i = 2; i <= n; ++i) {

vector<vector<int>> curr\_dp(2, vector<int>(3, 0));

// 遍历所有可能的前一天状态

for (int a = 0; a < 2; ++a) {

for (int l = 0; l < 3; ++l) {

int cnt = prev\_dp[a][l];

if (cnt == 0) continue; // 无此状态，跳过

// 1. 当天为'P'：连续迟到重置为0，缺勤次数不变

curr\_dp[a][0] = (curr\_dp[a][0] + cnt) % MOD;

// 2. 当天为'A'：仅允许a=0，缺勤次数变为1，连续迟到重置为0

if (a == 0) {

curr\_dp[1][0] = (curr\_dp[1][0] + cnt) % MOD;

}

// 3. 当天为'L'：连续迟到次数+1（不超过2），缺勤次数不变

if (l < 2) {

curr\_dp[a][l + 1] = (curr\_dp[a][l + 1] + cnt) % MOD;

}

}

}

prev\_dp = move(curr\_dp); // 更新前一天状态

}

// 累加所有有效状态的数量

int result = 0;

for (int a = 0; a < 2; ++a) {

for (int l = 0; l < 3; ++l) {

result = (result + prev\_dp[a][l]) % MOD;

}

}

return result;

}

};

代码解释

1、空间优化：

由于第i天的状态仅依赖第i-1天的状态，无需存储整个n×2×3的DP表，只需用两个2×3的二维数组（prev\_dp和curr\_dp）交替更新，将空间复杂度从O(n)降至O(1)。

2、状态转移细节：

对前一天的每个状态(a, l)，分别计算添加'P'、'A'、'L'后的新状态，并累加数量（取模1e9+7防止溢出）。

3、结果累加：

遍历最终状态（n天时的prev\_dp），累加所有有效状态的数量，得到总有效记录数。

复杂度分析

- 时间复杂度：O(n)，其中n为记录长度。需遍历n-1天，每天处理2×3=6种状态，总操作次数为O(n)。

- 空间复杂度：O(1)，仅使用固定大小的二维数组存储状态。

该方法通过动态规划高效追踪有效记录的状态转移，适合题目中n ≤ 1e5的约束，能在时间和空间上均保持最优性能。