📘 Student Attendance Record II – Full Documentation

📑 Table of Contents

1. Problem Statement

We are given an integer n, representing the length of a student's attendance record. The record consists of the characters:

- P (Present)
- A (Absent)
- L (Late)

A student is eligible for an attendance award if:

- They have fewer than 2 'A's in total, and
- They never have 3 or more consecutive 'L's

Return the number of valid attendance records of length n, modulo $10^9 + 7$.

2. Intuition

Rather than generating all sequences (which is exponential), we recognize patterns that allow us to build valid sequences dynamically using Dynamic Programming (DP).

We divide the problem into two sub-problems:

- Count sequences with 0 'A'
- Count sequences with exactly 1 'A'

These are added together to form the final answer.

3. Key Observations

- A sequence with more than one 'A' is invalid
- A sequence with "LLL" (three Lates) is invalid
- The idea is to build valid sequences from smaller ones using rules
- If we can count valid sequences without 'A', we can reuse that count to build longer valid sequences with 1 'A'

4. Approach

Step 1: Count all valid sequences with 0 'A'

- Use dynamic programming
- Let dp[i] = number of valid strings of length i without 'A' and without "LLL"

Recurrence:

dp[i] = dp[i-1] + dp[i-2] + dp[i-3]

Why?

- If the last character is P, append to dp[i-1]
- If last is L, ensure previous were not LL, so we use dp[i-2]
- If last is LL, we use dp[i-3]

Step 2: Count all valid sequences with exactly 1 'A'

- For each position i (from 0 to n-1), insert A at index i
- Multiply the number of valid strings on the left (dp[i]) with right (dp[n - 1 - i])

Final answer:

total = dp[n] + sum(dp[i] * dp[n - 1 - i] for i in range(n))

5. **Edge Cases**

- n = 1: Only three valid strings → P, L, A
- n = 2: Ensure correct count of combinations like LP, AP, etc.
- Very large n: Ensure algorithm runs within time constraints

6. **Complexity Analysis**

Time Complexity:

- O(n) — for building the dp array and for the loop inserting one 'A'

Space Complexity:

- O(n) — for the dp array

7. **Alternative Approaches**

- Recursive + Memoization: More intuitive but slower due to function call overhead
- Matrix Exponentiation: Can reduce DP to O(log n), but complex to implement
- Bottom-up Tabulation: Chosen here for its clarity and efficiency

8. **Algorithm**

- Initialize dp[0] = 1, dp[1] = 2, dp[2] = 4
- Use recurrence to fill dp[3] to dp[n]
- Start total = dp[n] (valid sequences with 0 'A')
- For each index i in 0 to n-1:
    - Add (dp[i] * dp[n - 1 - i]) to total
- Return total % MOD

9. **Flowchart**

[Start]
↓
[Initialize dp[0..n]]
↓
[Fill dp using recurrence dp[i] = dp[i-1] + dp[i-2] + dp[i-3]]
↓
[Set total = dp[n]]
↓
[For i in 0 to n-1:]
→ total += dp[i] * dp[n-1-i]
↓
[Return total % MOD]
↓
[End]

10. **Test Cases**

Input n	Expected Output	Explanation
1	3	P, L, A
2	8	All valid except "AA", "LLL"
3	19	(Check combinations manually)
10101	183236316	Large case, tests performance

11. Final Thoughts

This problem is a classic example of:

- Recognizing substructure and reuse via Dynamic Programming
- Handling combinatorial constraints
- Efficiently working with modulo arithmetic