Student Attendance Record II

Given a positive integer **n**, return the number of all possible attendance records with length n, which will be regarded as rewardable. The answer may be very large, return it after mod $10^9 + 7$.

A student attendance record is a string that only contains the following three characters:

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

A record is regarded as rewardable if it doesn't contain **more than one 'A' (absent)** or **more than two continuous 'L' (late)**.

Example 1:

```
Input: n = 2
Output: 8
Explanation:
There are 8 records with length 2 will be regarded as rewardable:
"PP" , "AP", "PA", "LP", "PL", "AL", "LA", "LL"
Only "AA" won't be regarded as rewardable owing to more than one absent times.
```

Note: The value of **n** won't exceed 100,000.

Solution 1

Let `f[i][j][k]` denote the # of valid sequences of length `i` where:

1. There can be at most `j` A's in the entire sequence.
2. There can be at most `k` **trailing** L's.

We give the recurrence in the following code, which should be self-explanatory, and the final answer is `f[n][1][2]`.

```
public int checkRecord(int n) {
    final int MOD = 1000000007;
    int[][][] f = new int[n + 1][2][3];

    f[0] = new int[][]{{1, 1, 1}, {1, 1, 1}};
    for (int i = 1; i <= n; i++)
        for (int j = 0; j < 2; j++)
            for (int k = 0; k < 3; k++) {
                int val = f[i - 1][j][2]; // ...P
                if (j > 0) val = (val + f[i - 1][j - 1][2]) % MOD; // ...A
                if (k > 0) val = (val + f[i - 1][j][k - 1]) % MOD; // ...L
                f[i][j][k] = val;
            }
    return f[n][1][2];
}
```

The runtime of this solution is clearly O(n), using linear space (which can be easily optimized to O(1) though). Now, let's see how to further improve the runtime.

In fact, if we treat `f[i][][]` and `f[i-1][][]` as two vectors, we can represent the recurrence of `f[i][j][k]` as follows:

```
f[i][0][0]   | 0 0 1 0 0 0 |   f[i-1][0][0]
f[i][0][1]   | 1 0 1 0 0 0 |   f[i-1][0][1]
f[i][0][2] = | 0 1 1 0 0 0 | * f[i-1][0][2]
f[i][1][0]   | 0 0 1 0 0 1 |   f[i-1][1][0]
f[i][1][1]   | 0 0 1 1 0 1 |   f[i-1][1][1]
f[i][1][2]   | 0 0 1 0 1 1 |   f[i-1][1][2]
```

Let `A` be the matrix above, then `f[n][][] = A^n * f[0][][]`, where `f[0][][] = [1 1 1 1 1 1]`. The point of this approach is that we can compute `A^n` using exponentiating by squaring (thanks to @StefanPochmann for the name correction), which will take O(6^3 * log n) = O(log n) time. Therefore, the runtime improves to O(log n), which suffices to handle the case for much larger `n`, say 10^18.
Update: The final answer is `f[n][1][2]`, which involves multiplying the last row of `A^n` and the column vector `[1 1 1 1 1 1]`. Interestingly, it is also equal to `A^(n+1)[5][2]` as the third column of `A` is just that vector. Credit to @StefanPochmann.

Java Code:

```
final int MOD = 1000000007;
final int M = 6;

int[][] mul(int[][] A, int[][] B) {
    int[][] C = new int[M][M];
    for (int i = 0; i < M; i++)
        for (int j = 0; j < M; j++)
            for (int k = 0; k < M; k++)
                C[i][j] = (int) ((C[i][j] + (long) A[i][k] * B[k][j]) % MOD);
    return C;
}


int[][] pow(int[][] A, int n) {
    int[][] res = new int[M][M];
    for (int i = 0; i < M; i++)
        res[i][i] = 1;
    while (n > 0) {
        if (n % 2 == 1)
            res = mul(res, A);
        A = mul(A, A);
        n /= 2;
    }
    return res;
}

public int checkRecord(int n) {
    int[][] A = {
            {0, 0, 1, 0, 0, 0},
            {1, 0, 1, 0, 0, 0},
            {0, 1, 1, 0, 0, 0},
            {0, 0, 1, 0, 0, 1},
            {0, 0, 1, 1, 0, 1},
            {0, 0, 1, 0, 1, 1},
    };
    return pow(A, n + 1)[5][2];
}
```

written by lixx2100 original link here

Solution 2

`dp[i]` the number of all possible attendance (without `'A'`) records with length `i` :

- end with `"P"` : `dp[i-1]`
- end with `"PL"` : `dp[i-2]`
- end with `"PLL"` : `dp[i-3]`
- end with `"LLL"` : is not allowed

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

the number of all possible attendance (with `'A'`) records with length `n` :
`∑dp[i] *dp[n-1-i]` `i = 0,1,...,n-1`

Time Complexity `O(n)`
Space Complexity `O(n)`

(In code `nums[i+1]` means `dp[i]`)

```
class Solution(object):
    def checkRecord(self, n):
        if n == 1:
            return 3
        if n == 0:
            return 0
        nums = [1, 1, 2]
        i = 2
        while i < n:
            nums.append((nums[i] + nums[i-1] + nums[i-2])% 1000000007)
            i += 1
        result = (nums[n] + nums[n-1] + nums[n-2]) % 1000000007
        for i in range(n):
            result += nums[i+1] * nums[n-i] % 1000000007
            result %= 1000000007
        return result
```

written by zym_ustc original link here

Solution 3

axly represents number of strings containing x A's and ending with y L's.

```
public class Solution {
    long M = 1000000007;
    public int checkRecord(int n) {
        long a0l0 = 1, a0l1 = 0, a0l2 = 0, a1l0 = 0, a1l1 = 0, a1l2 = 0;
        for (int i = 0; i <= n; i++) {
            long a0l0_ = (a0l0 + a0l1 + a0l2) % M;
            a0l2 = a0l1;
            a0l1 = a0l0;
            a0l0 = a0l0_;
            long a1l0_ = (a0l0 + a1l0 + a1l1 + a1l2) % M;
            a1l2 = a1l1;
            a1l1 = a1l0;
            a1l0 = a1l0_;
        }
        return (int) a1l0;
    }
}
```

written by vinod23 original link here