LeetCode 1578. 使绳子变成彩色的最短时间

Feijoa_Li

一、 题目描述

Alice 把 n 个气球排列在一根绳子上。给你一个下标从 0 开始的字符串 `colors`，其中 `colors[i]` 是第 i 个气球的颜色。

Alice 想要把绳子装扮成**五颜六色的**，且她不希望两个连续的气球涂着相同的颜色，所以她喊来 Bob 帮忙。Bob 可以从绳子上移除一些气球使绳子变成彩色。给你一个下标从 0 开始的整数数组 `neededTime`，其中 `neededTime[i]` 是 Bob 从绳子上移除第 i 个气球需要的时间（以秒为单位）。

返回 Bob 使绳子变成彩色需要的**最少时间**。

二、 解题思路

本题可以使用动态规划的方法解决。

状态定义： 定义 `dp[i][j]` 表示处理到第 i 个气球，且第 i 个气球的颜色为 j 时的最小花费时间。其中 j 的范围是 0-25（对应 26 个小写字母）和 26（表示移除该气球）。

初始状态：

- 第一个气球被移除：`dp[26] = neededTime[0]`
- 第一个气球保留：`dp[colors[0]-'a'] = 0`
- 其他状态初始化为无穷大

状态转移： 对于第 i 个气球（$i \geqslant 1$）：

- 如果当前气球被移除，那么前一个气球可以是任意状态，总时间增加 `neededTime[i]`
- 如果当前气球保留（颜色为 `col`），那么前一个气球不能是相同颜色，需要找到前一个气球所有非 `col` 颜色的最小花费

状态转移方程为：

$$\begin{cases} \texttt{dp[j] += neededTime[i]} & \text{对于所有 j（移除当前气球）} \\ \texttt{dp[col] = min(dp[col], Min)} & \text{其中 Min 是前一个状态中非 col 颜色的最小值} \end{cases}$$

三、 代码实现

```cpp
class Solution {
public: // 二维数组dp
const int inf = 1e9 + 7;
    int minCost(string colors, vector<int>& neededTime) {
        int n = neededTime.size();
        vector<vector<int>> dp(n,vector<int>(27,inf));
        dp[0][26] = neededTime[0];
        dp[0][colors[0] - 'a'] = 0;
        for(int i = 1 ; i < n ; i ++){
            int Min = inf;
            for(int j = 0 ; j <= 26 ; j ++) {
                if(j != colors[i] - 'a')Min = min(dp[i - 1][j] , Min)
                    ;
                dp[i][j] = min(dp[i][j] , dp[i - 1][j] + neededTime[i
                    ]);
            }
            dp[i][colors[i] - 'a'] = min(dp[i][colors[i] - 'a'] , Min
                );
        }

        return ranges::min(dp[n - 1]);
    }
};
```

```cpp
class Solution {
public: // 滚动数组优化dp
    const int inf = 1e9 + 7;
    int minCost(string colors, vector<int>& neededTime) {
        int n = neededTime.size();
        vector<int> dp(27, inf);
        // 初始状态：第一个气球
        dp[26] = neededTime[0];  // 移除第一个气球
        dp[colors[0] - 'a'] = 0; // 保留第一个气球

        for (int i = 1; i < n; i++) {
            int Min = inf;
            int col = colors[i] - 'a';
            // 找到前一个状态中非当前颜色的最小花费
            for (int j = 0; j <= 26; j++) {
                if (j != col) Min = min(dp[j], Min);
                // 移除当前气球，所有状态时间增加
                dp[j] += neededTime[i];
            }
            // 保留当前气球，只能从前一个非当前颜色的状态转移
            dp[col] = min(dp[col], Min);
        }

        return ranges::min(dp);
    }
```

```
};
```

四、 复杂度分析

- **时间复杂度：** $O(27 \times n)$，其中 n 是数组长度。对于每个气球需要遍历 27 个状态。
- **空间复杂度：** $O(27)$，只需要维护一个大小为 27 的状态数组。

五、 算法优化

实际上，本题还有更优的贪心解法：对于每一段连续相同颜色的气球，我们保留其中移除时间最大的那个，移除其他的气球。这样可以达到最优解。

```cpp
class Solution {
public: // 贪心
    const int inf = 1e9 + 7;
    int minCost(string colors, vector<int>& neededTime) {
        int res = 0, col = '␣', bef = 0;
        for (int i = 0; i < neededTime.size(); i++) {
            if (col == colors[i]) {
                res += min(bef, neededTime[i]);
                bef = max(bef, neededTime[i]);
            } else {
                col = colors[i];
                bef = neededTime[i];
            }
        }
        return res;
    }
};
```