1578. Minimum Time to Make Rope Colorful

Solved

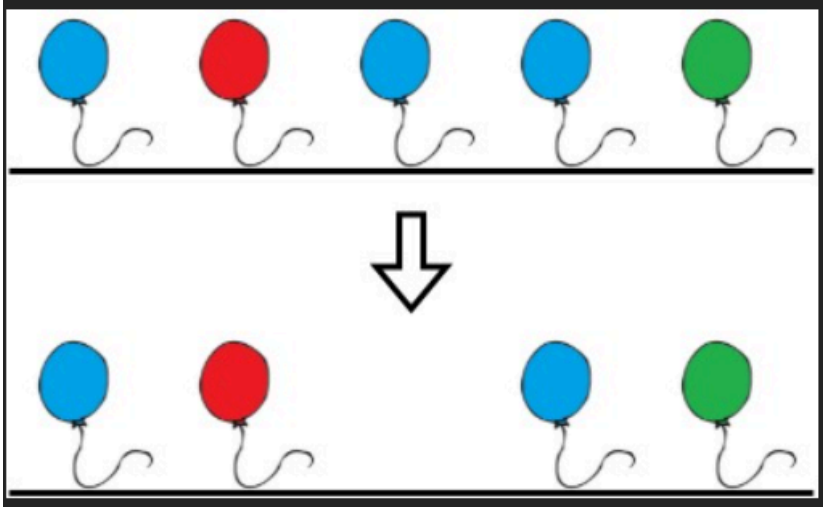
Medium | Topics | Companies | Hint

Alice has `n` balloons arranged on a rope. You are given a **0-indexed** string `colors` where `colors[i]` is the color of the `i^th` balloon.

Alice wants the rope to be **colorful**. She does not want **two consecutive balloons** to be of the same color, so she asks Bob for help. Bob can remove some balloons from the rope to make it **colorful**. You are given a **0-indexed** integer array `neededTime` where `neededTime[i]` is the time (in seconds) that Bob needs to remove the `i^th` balloon from the rope.

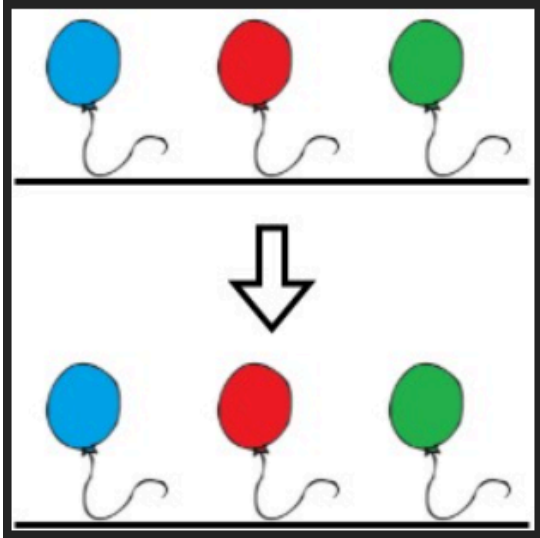
Return *the **minimum time** Bob needs to make the rope **colorful**.*

Example 1:



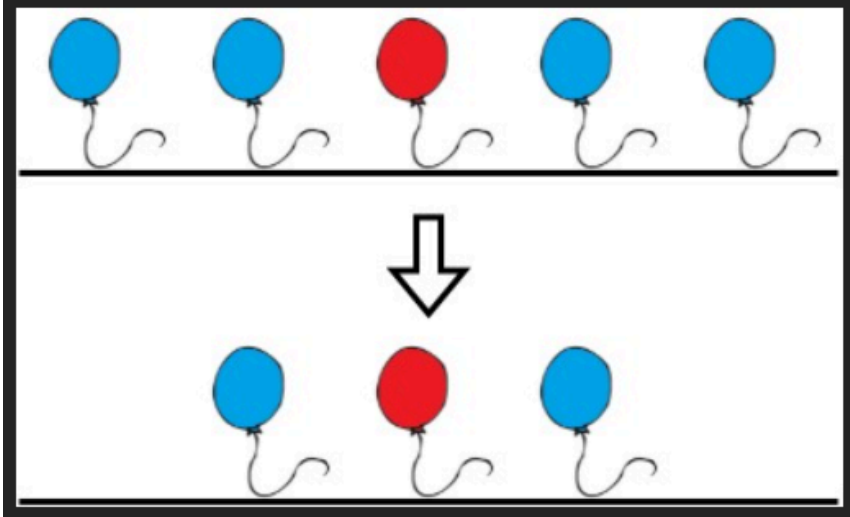
```
Input: colors = "abaac", neededTime = [1,2,3,4,5]
Output: 3
Explanation: In the above image, 'a' is blue, 'b' is red, and 'c' is
green.
Bob can remove the blue balloon at index 2. This takes 3 seconds.
There are no longer two consecutive balloons of the same color.
Total time = 3.
```

Example 2:

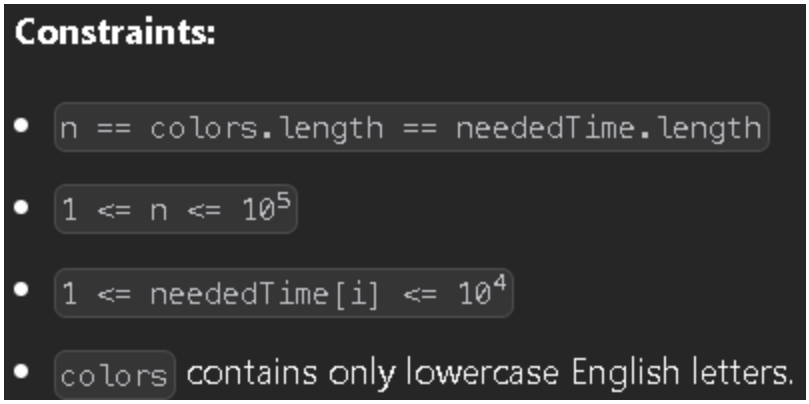


Input: colors = "abc", neededTime = [1,2,3]
Output: 0
Explanation: The rope is already colorful. Bob does not need to remove any balloons from the rope.

Example 3:



Input: colors = "aabaa", neededTime = [1,2,3,4,1]
Output: 2
Explanation: Bob will remove the balloons at indices 0 and 4. Each balloons takes 1 second to remove.
There are no longer two consecutive balloons of the same color. Total time = 1 + 1 = 2.



Python:

```
class Solution:
    def minCost(self, colors: str, neededTime: List[int]) -> int:
        totalTime = 0
        i = 0
        j = 0

        while i < len(neededTime) and j < len(neededTime):
            currTotal = 0
            currMax = 0

            while j < len(neededTime) and colors[i] == colors[j]:
                currTotal += neededTime[j]
                currMax = max(currMax, neededTime[j])
                j += 1

            totalTime += currTotal - currMax
            i = j

        return totalTime
```

JavaScript:

```
var minCost = function(colors, neededTime) {
    let totalTime = 0;
    let i = 0;
    let j = 0;

    while (i < neededTime.length && j < neededTime.length) {
        let currTotal = 0;
        let currMax = 0;
```

```
        while (j < neededTime.length && colors[i] === colors[j]) {
            currTotal += neededTime[j];
            currMax = Math.max(currMax, neededTime[j]);
            j++;
        }

        totalTime += currTotal - currMax;
        i = j;
    }

    return totalTime;
};
```

Java:

```
class Solution {
    public int minCost(String colors, int[] neededTime) {
        int totalTime = 0;
        int i = 0, j = 0;

        while (i < neededTime.length && j < neededTime.length) {
            int currTotal = 0, currMax = 0;

            while (j < neededTime.length && colors.charAt(i) == colors.charAt(j)) {
                currTotal += neededTime[j];
                currMax = Math.max(currMax, neededTime[j]);
                j++;
            }

            totalTime += currTotal - currMax;
            i = j;
        }
        return totalTime;
    }
}
```