Problem 1578: Minimum Time to Make Rope Colorful

Problem Information

Difficulty: Medium
Acceptance Rate: 0.00%
Paid Only: No

Problem Description

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

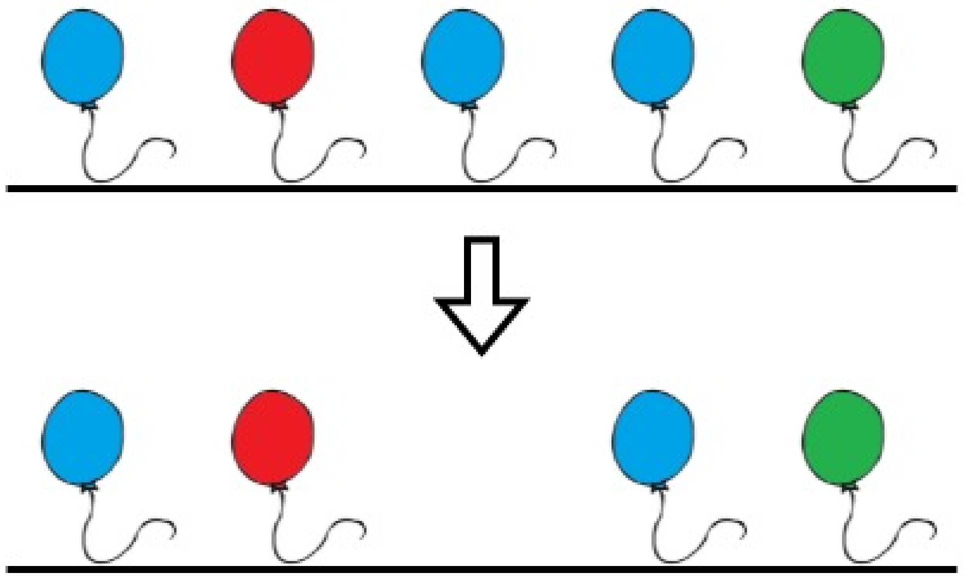
minimum time

Bob needs to make the rope

colorful

.

Example 1:



Input:

colors = "abaac", neededTime = [1,2,3,4,5]

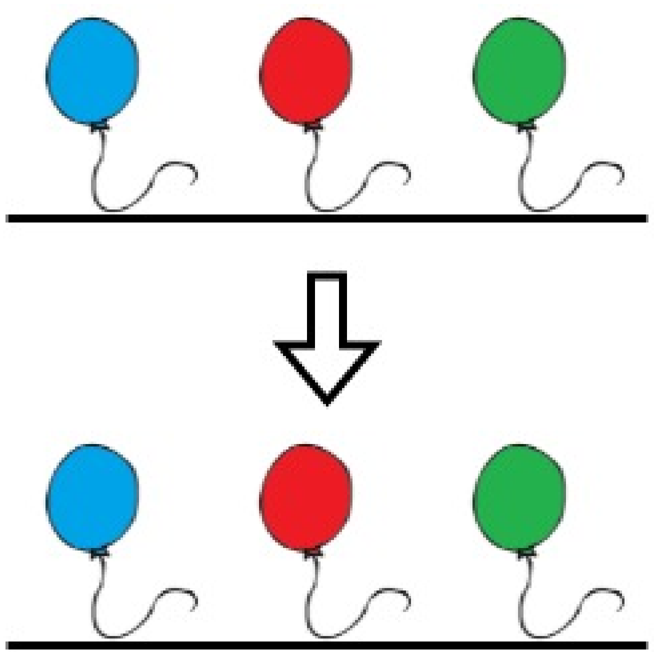
Output:

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]

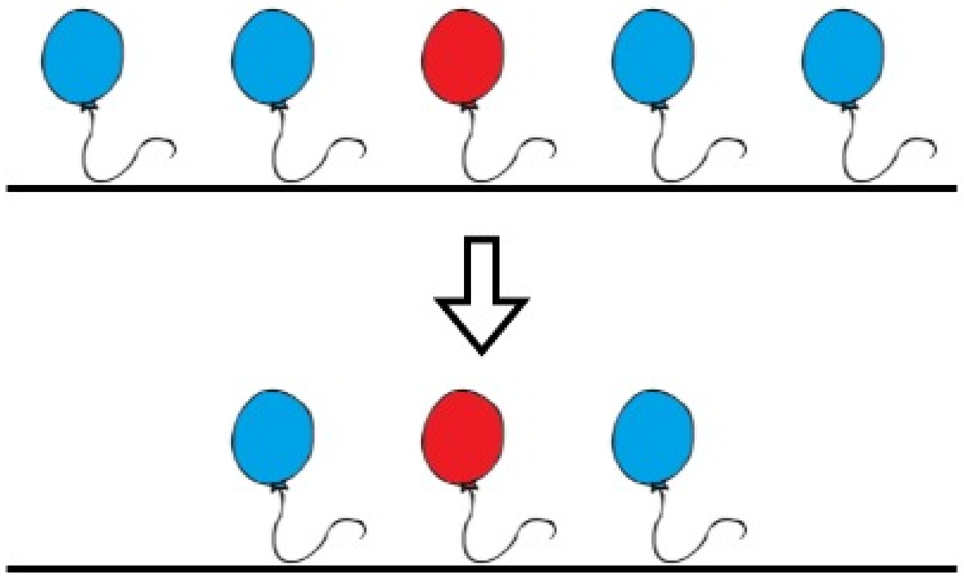
Output:

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.

Constraints:

n == colors.length == neededTime.length

1 <= n <= 10

5

1 <= neededTime[i] <= 10

4

colors

contains only lowercase English letters.

Code Snippets

C++:

```
class Solution {
public:
int minCost(string colors, vector<int>& neededTime) {

}
};
```

Java:

```
class Solution {
public int minCost(String colors, int[] neededTime) {

}
}
```

Python3:

```
class Solution:
def minCost(self, colors: str, neededTime: List[int]) -> int:
```

Python:

```
class Solution(object):
def minCost(self, colors, neededTime):
"""
:type colors: str
:type neededTime: List[int]
:rtype: int
"""
```

JavaScript:

```
/**
* @param {string} colors
* @param {number[]} neededTime
* @return {number}
*/
var minCost = function(colors, neededTime) {

};
```

TypeScript:

```
function minCost(colors: string, neededTime: number[]): number {

};
```

C#:

```
public class Solution {
public int MinCost(string colors, int[] neededTime) {

}
}
```

C:

```
int minCost(char* colors, int* neededTime, int neededTimeSize) {

}
```

Go:

```
func minCost(colors string, neededTime []int) int {

}
```

Kotlin:

```
class Solution {
fun minCost(colors: String, neededTime: IntArray): Int {

}
}
```

Swift:

```
class Solution {
func minCost(_ colors: String, _ neededTime: [Int]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn min_cost(colors: String, needed_time: Vec<i32>) -> i32 {

}
}
```

Ruby:

```
# @param {String} colors
# @param {Integer[]} needed_time
# @return {Integer}
def min_cost(colors, needed_time)

end
```

PHP:

```
class Solution {

/**
* @param String $colors
* @param Integer[] $neededTime
* @return Integer
*/
function minCost($colors, $neededTime) {

}
}
```

Dart:

```
class Solution {
int minCost(String colors, List<int> neededTime) {

}

}
```

Scala:

```
object Solution {
def minCost(colors: String, neededTime: Array[Int]): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec min_cost(colors :: String.t, needed_time :: [integer]) :: integer
def min_cost(colors, needed_time) do

end
end
```

Erlang:

```
-spec min_cost(Colors :: unicode:unicode_binary(), NeededTime :: [integer()])
-> integer().
min_cost(Colors, NeededTime) ->
.
```

Racket:

```
(define/contract (min-cost colors neededTime)
(-> string? (listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Minimum Time to Make Rope Colorful
* Difficulty: Medium
* Tags: array, string, dp, greedy
```

```
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Solution {
public:
int minCost(string colors, vector<int>& neededTime) {

}
};
```

Java Solution:

```
/**
* Problem: Minimum Time to Make Rope Colorful
* Difficulty: Medium
* Tags: array, string, dp, greedy
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Solution {
public int minCost(String colors, int[] neededTime) {

}
}
```

Python3 Solution:

```
"""
Problem: Minimum Time to Make Rope Colorful
Difficulty: Medium
Tags: array, string, dp, greedy

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) or O(n * m) for DP table
```

```
"""

class Solution:
def minCost(self, colors: str, neededTime: List[int]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def minCost(self, colors, neededTime):
"""
:type colors: str
:type neededTime: List[int]
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Minimum Time to Make Rope Colorful
* Difficulty: Medium
* Tags: array, string, dp, greedy
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

/**
* @param {string} colors
* @param {number[]} neededTime
* @return {number}
*/
var minCost = function(colors, neededTime) {

};
```

TypeScript Solution:

```
/**
* Problem: Minimum Time to Make Rope Colorful
* Difficulty: Medium
* Tags: array, string, dp, greedy
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

function minCost(colors: string, neededTime: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Minimum Time to Make Rope Colorful
* Difficulty: Medium
* Tags: array, string, dp, greedy
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

public class Solution {
public int MinCost(string colors, int[] neededTime) {

}
}
```

C Solution:

```
/*
* Problem: Minimum Time to Make Rope Colorful
* Difficulty: Medium
* Tags: array, string, dp, greedy
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
```

```
*/

int minCost(char* colors, int* neededTime, int neededTimeSize) {

}
```

Go Solution:

```
// Problem: Minimum Time to Make Rope Colorful
// Difficulty: Medium
// Tags: array, string, dp, greedy
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

func minCost(colors string, neededTime []int) int {

}
```

Kotlin Solution:

```
class Solution {
fun minCost(colors: String, neededTime: IntArray): Int {

}
}
```

Swift Solution:

```
class Solution {
func minCost(_ colors: String, _ neededTime: [Int]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Minimum Time to Make Rope Colorful
// Difficulty: Medium
// Tags: array, string, dp, greedy
```

```
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

impl Solution {
pub fn min_cost(colors: String, needed_time: Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {String} colors
# @param {Integer[]} needed_time
# @return {Integer}
def min_cost(colors, needed_time)

end
```

PHP Solution:

```
class Solution {

/**
* @param String $colors
* @param Integer[] $neededTime
* @return Integer
*/
function minCost($colors, $neededTime) {

}
}
```

Dart Solution:

```
class Solution {
int minCost(String colors, List<int> neededTime) {

}
}
```

Scala Solution:

```
object Solution {
def minCost(colors: String, neededTime: Array[Int]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec min_cost(colors :: String.t, needed_time :: [integer]) :: integer
def min_cost(colors, needed_time) do

end
end
```

Erlang Solution:

```
-spec min_cost(Colors :: unicode:unicode_binary(), NeededTime :: [integer()])
-> integer().
min_cost(Colors, NeededTime) ->
.
```

Racket Solution:

```
(define/contract (min-cost colors neededTime)
(-> string? (listof exact-integer?) exact-integer?)
)
```