Problem 2399: Check Distances Between Same Letters

Problem Information

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

Problem Description

You are given a

0-indexed

string

s

consisting of only lowercase English letters, where each letter in

s

appears

exactly

twice

. You are also given a

0-indexed

integer array

distance

of length

26

.

Each letter in the alphabet is numbered from

0

to

25

(i.e.

'a' -> 0

,

'b' -> 1

,

'c' -> 2

, ... ,

'z' -> 25

).

In a

well-spaced

string, the number of letters between the two occurrences of the i

th

letter is

distance[i]

. If the

i

th

letter does not appear in

s

, then

distance[i]

can be

ignored

.

Return

true

if

s

is a

well-spaced

string, otherwise return

false

.

Example 1:

Input:

s = "abaccb", distance = [1,3,0,5,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]

Output:

true

Explanation:

- 'a' appears at indices 0 and 2 so it satisfies distance[0] = 1. - 'b' appears at indices 1 and 5 so it satisfies distance[1] = 3. - 'c' appears at indices 3 and 4 so it satisfies distance[2] = 0. Note that distance[3] = 5, but since 'd' does not appear in s, it can be ignored. Return true because s is a well-spaced string.

Example 2:

Input:

s = "aa", distance = [1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]

Output:

false

Explanation:

- 'a' appears at indices 0 and 1 so there are zero letters between them. Because distance[0] = 1, s is not a well-spaced string.

Constraints:

2 <= s.length <= 52

s

consists only of lowercase English letters.

Each letter appears in

s

exactly twice.

distance.length == 26

0 <= distance[i] <= 50

Code Snippets

C++:

```
class Solution {
public:
bool checkDistances(string s, vector<int>& distance) {

}
};
```

Java:

```
class Solution {
public boolean checkDistances(String s, int[] distance) {

}
}
```

Python3:

```
class Solution:
def checkDistances(self, s: str, distance: List[int]) -> bool:
```

Python:

```
class Solution(object):
def checkDistances(self, s, distance):
"""
:type s: str
:type distance: List[int]
:rtype: bool
"""
```

JavaScript:

```
/**
* @param {string} s
* @param {number[]} distance
* @return {boolean}
*/
var checkDistances = function(s, distance) {

};
```

TypeScript:

```
function checkDistances(s: string, distance: number[]): boolean {

};
```

C#:

```
public class Solution {
public bool CheckDistances(string s, int[] distance) {

}
}
```

C:

```
bool checkDistances(char* s, int* distance, int distanceSize) {

}
```

Go:

```
func checkDistances(s string, distance []int) bool {

}
```

Kotlin:

```
class Solution {
fun checkDistances(s: String, distance: IntArray): Boolean {

}
}
```

Swift:

```
class Solution {
func checkDistances(_ s: String, _ distance: [Int]) -> Bool {

}
}
```

Rust:

```
impl Solution {
pub fn check_distances(s: String, distance: Vec<i32>) -> bool {

}
}
```

Ruby:

```
# @param {String} s
# @param {Integer[]} distance
# @return {Boolean}
def check_distances(s, distance)

end
```

PHP:

```
class Solution {

/**
* @param String $s
* @param Integer[] $distance
* @return Boolean
*/
function checkDistances($s, $distance) {

}
}
```

Dart:

```
class Solution {
bool checkDistances(String s, List<int> distance) {

}
}
```

Scala:

```
object Solution {
def checkDistances(s: String, distance: Array[Int]): Boolean = {

}
}
```

Elixir:

```
defmodule Solution do
@spec check_distances(s :: String.t, distance :: [integer]) :: boolean
def check_distances(s, distance) do

end
end
```

Erlang:

```
-spec check_distances(S :: unicode:unicode_binary(), Distance :: [integer()])
-> boolean().
check_distances(S, Distance) ->
.
```

Racket:

```
(define/contract (check-distances s distance)
(-> string? (listof exact-integer?) boolean?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Check Distances Between Same Letters
* Difficulty: Easy
* Tags: array, string, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

class Solution {
public:
bool checkDistances(string s, vector<int>& distance) {

}
};
```

Java Solution:

```
/**
* Problem: Check Distances Between Same Letters
* Difficulty: Easy
* Tags: array, string, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

class Solution {
public boolean checkDistances(String s, int[] distance) {

}
}
```

Python3 Solution:

```
"""
Problem: Check Distances Between Same Letters
Difficulty: Easy
Tags: array, string, hash

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) for hash map
"""

class Solution:
def checkDistances(self, s: str, distance: List[int]) -> bool:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def checkDistances(self, s, distance):
"""
:type s: str
:type distance: List[int]
:rtype: bool
"""
```

JavaScript Solution:

```
/**
* Problem: Check Distances Between Same Letters
* Difficulty: Easy
* Tags: array, string, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/
```

```
/**
* @param {string} s
* @param {number[]} distance
* @return {boolean}
*/
var checkDistances = function(s, distance) {

};
```

TypeScript Solution:

```
/**
* Problem: Check Distances Between Same Letters
* Difficulty: Easy
* Tags: array, string, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

function checkDistances(s: string, distance: number[]): boolean {

};
```

C# Solution:

```
/*
* Problem: Check Distances Between Same Letters
* Difficulty: Easy
* Tags: array, string, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

public class Solution {
public bool CheckDistances(string s, int[] distance) {

}
}
```

C Solution:

```c
/*
* Problem: Check Distances Between Same Letters
* Difficulty: Easy
* Tags: array, string, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

bool checkDistances(char* s, int* distance, int distanceSize) {

}
```

Go Solution:

```go
// Problem: Check Distances Between Same Letters
// Difficulty: Easy
// Tags: array, string, hash
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

func checkDistances(s string, distance []int) bool {

}
```

Kotlin Solution:

```kotlin
class Solution {
fun checkDistances(s: String, distance: IntArray): Boolean {

}
}
```

Swift Solution:

```
class Solution {
func checkDistances(_ s: String, _ distance: [Int]) -> Bool {

}
}
```

Rust Solution:

```
// Problem: Check Distances Between Same Letters
// Difficulty: Easy
// Tags: array, string, hash
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

impl Solution {
pub fn check_distances(s: String, distance: Vec<i32>) -> bool {

}
}
```

Ruby Solution:

```
# @param {String} s
# @param {Integer[]} distance
# @return {Boolean}
def check_distances(s, distance)

end
```

PHP Solution:

```
class Solution {

/**
* @param String $s
* @param Integer[] $distance
* @return Boolean
*/
function checkDistances($s, $distance) {
}
}
```

Dart Solution:

```
class Solution {
bool checkDistances(String s, List<int> distance) {

}
}
```

Scala Solution:

```
object Solution {
def checkDistances(s: String, distance: Array[Int]): Boolean = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec check_distances(s :: String.t, distance :: [integer]) :: boolean
def check_distances(s, distance) do

end
end
```

Erlang Solution:

```
-spec check_distances(S :: unicode:unicode_binary(), Distance :: [integer()])
-> boolean().
check_distances(S, Distance) ->
.
```

Racket Solution:

```
(define/contract (check-distances s distance)
(-> string? (listof exact-integer?) boolean?)
)
```