Problem 2124: Check if All A's Appears Before All B's

Problem Information

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

Problem Description

Given a string

s

consisting of

only

the characters

'a'

and

'b'

, return

true

if

every

'a'

appears before

every

'b'

in the string

. Otherwise, return

false

.

Example 1:

Input:

s = "aaabbb"

Output:

true

Explanation:

The 'a's are at indices 0, 1, and 2, while the 'b's are at indices 3, 4, and 5. Hence, every 'a' appears before every 'b' and we return true.

Example 2:

Input:

s = "abab"

Output:

false

Explanation:

There is an 'a' at index 2 and a 'b' at index 1. Hence, not every 'a' appears before every 'b' and we return false.

Example 3:

Input:

s = "bbb"

Output:

true

Explanation:

There are no 'a's, hence, every 'a' appears before every 'b' and we return true.

Constraints:

1 <= s.length <= 100

s[i]

is either

'a'

or

'b'

.

Code Snippets

C++:

```
class Solution {
public:
bool checkString(string s) {

}
};
```

Java:

```
class Solution {
public boolean checkString(String s) {

}
}
```

Python3:

```
class Solution:
def checkString(self, s: str) -> bool:
```

Python:

```
class Solution(object):
def checkString(self, s):
"""
:type s: str
:rtype: bool
"""
```

JavaScript:

```
/**
* @param {string} s
* @return {boolean}
*/
var checkString = function(s) {

};
```

TypeScript:

```
function checkString(s: string): boolean {

};
```

C#:

```
public class Solution {
public bool CheckString(string s) {

}
}
```

C:

```
bool checkString(char* s) {

}
```

Go:

```
func checkString(s string) bool {

}
```

Kotlin:

```
class Solution {
fun checkString(s: String): Boolean {

}
}
```

Swift:

```
class Solution {
func checkString(_ s: String) -> Bool {

}
}
```

Rust:

```
impl Solution {
pub fn check_string(s: String) -> bool {

}
}
```

Ruby:

```
# @param {String} s
# @return {Boolean}
def check_string(s)

end
```

PHP:

```
class Solution {

/**
* @param String $s
* @return Boolean
*/
function checkString($s) {

}
}
```

Dart:

```
class Solution {
bool checkString(String s) {

}
}
```

Scala:

```
object Solution {
def checkString(s: String): Boolean = {

}
}
```

Elixir:

```
defmodule Solution do
@spec check_string(s :: String.t) :: boolean
def check_string(s) do

end
end
```

Erlang:

```
-spec check_string(S :: unicode:unicode_binary()) -> boolean().
check_string(S) ->
.
```

Racket:

```
(define/contract (check-string s)
(-> string? boolean?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Check if All A's Appears Before All B's
* Difficulty: Easy
* Tags: string
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public:
bool checkString(string s) {

}
};
```

Java Solution:

```
/**
* Problem: Check if All A's Appears Before All B's
* Difficulty: Easy
* Tags: string
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public boolean checkString(String s) {

}
}
```

Python3 Solution:

```
"""
Problem: Check if All A's Appears Before All B's
Difficulty: Easy
Tags: string

Approach: String manipulation with hash map or two pointers
Time Complexity: O(n) or O(n log n)
Space Complexity: O(1) to O(n) depending on approach
"""

class Solution:
def checkString(self, s: str) -> bool:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def checkString(self, s):
"""
:type s: str
:rtype: bool
```

```
"""
```

JavaScript Solution:

```
/**
* Problem: Check if All A's Appears Before All B's
* Difficulty: Easy
* Tags: string
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

/**
* @param {string} s
* @return {boolean}
*/
var checkString = function(s) {

};
```

TypeScript Solution:

```
/**
* Problem: Check if All A's Appears Before All B's
* Difficulty: Easy
* Tags: string
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

function checkString(s: string): boolean {

};
```

C# Solution:

```
/*
* Problem: Check if All A's Appears Before All B's
* Difficulty: Easy
* Tags: string
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

public class Solution {
public bool CheckString(string s) {

}
}
```

C Solution:

```
/*
* Problem: Check if All A's Appears Before All B's
* Difficulty: Easy
* Tags: string
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

bool checkString(char* s) {

}
```

Go Solution:

```
// Problem: Check if All A's Appears Before All B's
// Difficulty: Easy
// Tags: string
//
// Approach: String manipulation with hash map or two pointers
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach
```

```
func checkString(s string) bool {

}
```

Kotlin Solution:

```
class Solution {
fun checkString(s: String): Boolean {

}
}
```

Swift Solution:

```
class Solution {
func checkString(_ s: String) -> Bool {

}
}
```

Rust Solution:

```
// Problem: Check if All A's Appears Before All B's
// Difficulty: Easy
// Tags: string
//
// Approach: String manipulation with hash map or two pointers
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

impl Solution {
pub fn check_string(s: String) -> bool {

}
}
```

Ruby Solution:

```
# @param {String} s
# @return {Boolean}
def check_string(s)
```

```
end
```

PHP Solution:

```
class Solution {

/**
* @param String $s
* @return Boolean
*/
function checkString($s) {

}
}
```

Dart Solution:

```
class Solution {
bool checkString(String s) {

}
}
```

Scala Solution:

```
object Solution {
def checkString(s: String): Boolean = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec check_string(s :: String.t) :: boolean
def check_string(s) do

end
end
```

Erlang Solution:

```
-spec check_string(S :: unicode:unicode_binary()) -> boolean().
check_string(S) ->
.
```

Racket Solution:

```
(define/contract (check-string s)
(-> string? boolean?)
)
```