Problem 859: Buddy Strings

Problem Information

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

Problem Description

Given two strings

s

and

goal

, return

true

if you can swap two letters in

s

so the result is equal to

goal

, otherwise, return

false

.

Swapping letters is defined as taking two indices

i

and

j

(0-indexed) such that

i != j

and swapping the characters at

s[i]

and

s[j]

.

For example, swapping at indices

0

and

2

in

"abcd"

results in

"cbad"

.

Example 1:

Input:

s = "ab", goal = "ba"

Output:

true

Explanation:

You can swap s[0] = 'a' and s[1] = 'b' to get "ba", which is equal to goal.

Example 2:

Input:

s = "ab", goal = "ab"

Output:

false

Explanation:

The only letters you can swap are s[0] = 'a' and s[1] = 'b', which results in "ba" != goal.

Example 3:

Input:

s = "aa", goal = "aa"

Output:

true

Explanation:

You can swap s[0] = 'a' and s[1] = 'a' to get "aa", which is equal to goal.

Constraints:

1 <= s.length, goal.length <= 2 * 10

4

s

and

goal

consist of lowercase letters.

Code Snippets

C++:

```
class Solution {
public:
bool buddyStrings(string s, string goal) {

}
};
```

Java:

```
class Solution {
public boolean buddyStrings(String s, String goal) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function buddyStrings(s: string, goal: string): boolean {

};
```

C#:

```
public class Solution {
public bool BuddyStrings(string s, string goal) {

}
}
```

C:

```
bool buddyStrings(char* s, char* goal) {

}
```

Go:

```
func buddyStrings(s string, goal string) bool {

}
```

Kotlin:

```
class Solution {
fun buddyStrings(s: String, goal: String): Boolean {

}
}
```

Swift:

```
class Solution {
func buddyStrings(_ s: String, _ goal: String) -> Bool {

}
}
```

Rust:

```
impl Solution {
pub fn buddy_strings(s: String, goal: String) -> bool {

}
}
```

Ruby:

```
# @param {String} s
# @param {String} goal
# @return {Boolean}
def buddy_strings(s, goal)

end
```

PHP:

```
class Solution {
```

```
/**
* @param String $s
* @param String $goal
* @return Boolean
*/
function buddyStrings($s, $goal) {

}
}
```

Dart:

```
class Solution {
bool buddyStrings(String s, String goal) {

}
}
```

Scala:

```
object Solution {
def buddyStrings(s: String, goal: String): Boolean = {

}
}
```

Elixir:

```
defmodule Solution do
@spec buddy_strings(s :: String.t, goal :: String.t) :: boolean
def buddy_strings(s, goal) do

end
end
```

Erlang:

```
-spec buddy_strings(S :: unicode:unicode_binary(), Goal ::
unicode:unicode_binary()) -> boolean().
buddy_strings(S, Goal) ->
.
```

Racket:

```
(define/contract (buddy-strings s goal)
(-> string? string? boolean?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Buddy Strings
* Difficulty: Easy
* Tags: string, hash
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

class Solution {
public:
bool buddyStrings(string s, string goal) {

}
};
```

Java Solution:

```
/**
* Problem: Buddy Strings
* Difficulty: Easy
* Tags: string, hash
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

class Solution {
public boolean buddyStrings(String s, String goal) {
}
}
```

Python3 Solution:

```
"""
Problem: Buddy Strings
Difficulty: Easy
Tags: string, hash

Approach: String manipulation with hash map or two pointers
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) for hash map
"""

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Buddy Strings
* Difficulty: Easy
* Tags: string, hash
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
```

```
*/

/**
* @param {string} s
* @param {string} goal
* @return {boolean}
*/
var buddyStrings = function(s, goal) {

};
```

TypeScript Solution:

```
/**
* Problem: Buddy Strings
* Difficulty: Easy
* Tags: string, hash
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

function buddyStrings(s: string, goal: string): boolean {

};
```

C# Solution:

```
/*
* Problem: Buddy Strings
* Difficulty: Easy
* Tags: string, hash
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

public class Solution {
public bool BuddyStrings(string s, string goal) {
}
}
```

C Solution:

```
/*
* Problem: Buddy Strings
* Difficulty: Easy
* Tags: string, hash
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

bool buddyStrings(char* s, char* goal) {

}
```

Go Solution:

```
// Problem: Buddy Strings
// Difficulty: Easy
// Tags: string, hash
//
// Approach: String manipulation with hash map or two pointers
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

func buddyStrings(s string, goal string) bool {

}
```

Kotlin Solution:

```
class Solution {
fun buddyStrings(s: String, goal: String): Boolean {

}
}
```

Swift Solution:

```
class Solution {
func buddyStrings(_ s: String, _ goal: String) -> Bool {

}
}
```

Rust Solution:

```
// Problem: Buddy Strings
// Difficulty: Easy
// Tags: string, hash
//
// Approach: String manipulation with hash map or two pointers
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

impl Solution {
pub fn buddy_strings(s: String, goal: String) -> bool {

}
}
```

Ruby Solution:

```
# @param {String} s
# @param {String} goal
# @return {Boolean}
def buddy_strings(s, goal)

end
```

PHP Solution:

```
class Solution {

/**
* @param String $s
* @param String $goal
* @return Boolean
*/
function buddyStrings($s, $goal) {

}
}
```

Dart Solution:

```
class Solution {
bool buddyStrings(String s, String goal) {

}
}
```

Scala Solution:

```
object Solution {
def buddyStrings(s: String, goal: String): Boolean = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec buddy_strings(s :: String.t, goal :: String.t) :: boolean
def buddy_strings(s, goal) do

end
end
```

Erlang Solution:

```
-spec buddy_strings(S :: unicode:unicode_binary(), Goal ::
unicode:unicode_binary()) -> boolean().
buddy_strings(S, Goal) ->
.
```

Racket Solution:

```
(define/contract (buddy-strings s goal)
(-> string? string? boolean?)

)
```