Problem 844: Backspace String Compare

Problem Information

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

Problem Description

Given two strings

s

and

t

, return

true

if they are equal when both are typed into empty text editors

.

'#'

means a backspace character.

Note that after backspacing an empty text, the text will continue empty.

Example 1:

Input:

s = "ab#c", t = "ad#c"

Output:

true

Explanation:

Both s and t become "ac".

Example 2:

Input:

s = "ab##", t = "c#d#"

Output:

true

Explanation:

Both s and t become "".

Example 3:

Input:

s = "a#c", t = "b"

Output:

false

Explanation:

s becomes "c" while t becomes "b".

Constraints:

1 <= s.length, t.length <= 200

s

and

t

only contain lowercase letters and

'#'

characters.

Follow up:

Can you solve it in

O(n)

time and

O(1)

space?

Code Snippets

C++:

```
class Solution {
public:
bool backspaceCompare(string s, string t) {

}
};
```

Java:

```
class Solution {
public boolean backspaceCompare(String s, String t) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function backspaceCompare(s: string, t: string): boolean {

};
```

C#:

```
public class Solution {
public bool BackspaceCompare(string s, string t) {
```

```
}
}
```

C:

```
bool backspaceCompare(char* s, char* t) {

}
```

Go:

```
func backspaceCompare(s string, t string) bool {

}
```

Kotlin:

```
class Solution {
fun backspaceCompare(s: String, t: String): Boolean {

}
}
```

Swift:

```
class Solution {
func backspaceCompare(_ s: String, _ t: String) -> Bool {

}
}
```

Rust:

```
impl Solution {
pub fn backspace_compare(s: String, t: String) -> bool {

}
}
```

Ruby:

```
# @param {String} s
# @param {String} t
# @return {Boolean}
def backspace_compare(s, t)

end
```

PHP:

```
class Solution {

/**
* @param String $s
* @param String $t
* @return Boolean
*/
function backspaceCompare($s, $t) {

}
}
```

Dart:

```
class Solution {
bool backspaceCompare(String s, String t) {

}
}
```

Scala:

```
object Solution {
def backspaceCompare(s: String, t: String): Boolean = {

}
}
```

Elixir:

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

```
end
end
```

Erlang:

```
-spec backspace_compare(S :: unicode:unicode_binary(), T ::
unicode:unicode_binary()) -> boolean().
backspace_compare(S, T) ->
.
```

Racket:

```
(define/contract (backspace-compare s t)
(-> string? string? boolean?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Backspace String Compare
* Difficulty: Easy
* Tags: array, string, stack
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public:
bool backspaceCompare(string s, string t) {

}
};
```

Java Solution:

```
/**
* Problem: Backspace String Compare
* Difficulty: Easy
* Tags: array, string, stack
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public boolean backspaceCompare(String s, String t) {

}
}
```

Python3 Solution:

```
"""
Problem: Backspace String Compare
Difficulty: Easy
Tags: array, string, stack

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(1) to O(n) depending on approach
"""

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Backspace String Compare
* Difficulty: Easy
* Tags: array, string, stack
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

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

};
```

TypeScript Solution:

```
/**
* Problem: Backspace String Compare
* Difficulty: Easy
* Tags: array, string, stack
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

function backspaceCompare(s: string, t: string): boolean {

};
```

C# Solution:

```
/*
* Problem: Backspace String Compare
* Difficulty: Easy
```

```
* Tags: array, string, stack
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

public class Solution {
public bool BackspaceCompare(string s, string t) {

}
}
```

C Solution:

```
/*
* Problem: Backspace String Compare
* Difficulty: Easy
* Tags: array, string, stack
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

bool backspaceCompare(char* s, char* t) {

}
```

Go Solution:

```
// Problem: Backspace String Compare
// Difficulty: Easy
// Tags: array, string, stack
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

func backspaceCompare(s string, t string) bool {
```

```
}
```

Kotlin Solution:

```
class Solution {
fun backspaceCompare(s: String, t: String): Boolean {

}
}
```

Swift Solution:

```
class Solution {
func backspaceCompare(_ s: String, _ t: String) -> Bool {

}
}
```

Rust Solution:

```
// Problem: Backspace String Compare
// Difficulty: Easy
// Tags: array, string, stack
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

impl Solution {
pub fn backspace_compare(s: String, t: String) -> bool {

}
}
```

Ruby Solution:

```
# @param {String} s
# @param {String} t
# @return {Boolean}
def backspace_compare(s, t)
```

```
end
```

PHP Solution:

```
class Solution {

/**
* @param String $s
* @param String $t
* @return Boolean
*/
function backspaceCompare($s, $t) {

}
}
```

Dart Solution:

```
class Solution {
bool backspaceCompare(String s, String t) {

}
}
```

Scala Solution:

```
object Solution {
def backspaceCompare(s: String, t: String): Boolean = {

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

```
-spec backspace_compare(S :: unicode:unicode_binary(), T ::
unicode:unicode_binary()) -> boolean().
backspace_compare(S, T) ->
.
```

Racket Solution:

```
(define/contract (backspace-compare s t)
(-> string? string? boolean?)
)
```