

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:

$1 \leq s.length, t.length \leq 200$

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 {
        var i = s.length - 1
        var j = t.length - 1
        var skipS = 0
        var skipT = 0

        while (i >= 0 || j >= 0) {
            while (i >= 0) {
                if (s[i] == '#') {
                    skipS++
                    i--
                } else if (skipS == 0) {
                    break
                } else {
                    i--
                    skipS--
                }
            }

            while (j >= 0) {
                if (t[j] == '#') {
                    skipT++
                    j--
                } else if (skipT == 0) {
                    break
                } else {
                    j--
                    skipT--
                }
            }

            if (i < 0 || j < 0) {
                return i == j
            }

            if (s[i] != t[j]) {
                return false
            }

            i--
            j--
        }

        return true
    }
}
```

Swift Solution:

```
class Solution {
    func backspaceCompare(_ s: String, _ t: String) -> Bool {
        let sArr = Array(s)
        let tArr = Array(t)
        var i = 0
        var j = 0
        while i < sArr.count || j < tArr.count {
            if i < sArr.count && sArr[i] == "#" {
                i += 1
                continue
            }
            if j < tArr.count && tArr[j] == "#" {
                j += 1
                continue
            }
            if i < sArr.count && j < tArr.count && sArr[i] == tArr[j] {
                i += 1
                j += 1
            } else {
                return false
            }
        }
        return true
    }
}
```

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 {
        let mut s_index = 0;
        let mut t_index = 0;

        while s_index < s.len() || t_index < t.len() {
            if s_index < s.len() {
                if s[s_index] == '#' {
                    s_index += 1;
                    continue;
                }
                let mut skip_s = 0;
                while s_index + skip_s < s.len() {
                    if s[s_index + skip_s] == '#' {
                        skip_s += 1;
                    } else {
                        break;
                    }
                }
                s_index += skip_s;
            }

            if t_index < t.len() {
                if t[t_index] == '#' {
                    t_index += 1;
                    continue;
                }
                let mut skip_t = 0;
                while t_index + skip_t < t.len() {
                    if t[t_index + skip_t] == '#' {
                        skip_t += 1;
                    } else {
                        break;
                    }
                }
                t_index += skip_t;
            }

            if s_index < s.len() && t_index < t.len() {
                if s[s_index] != t[t_index] {
                    return false;
                }
                s_index += 1;
                t_index += 1;
            }
        }

        s_index == s.len() && t_index == t.len()
    }
}
```

Ruby Solution:

```
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?)  
  )
```