Problem 242: Valid Anagram

Problem Information

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

Problem Description

Given two strings

s

and

t

, return

true

if

t

is an

anagram

of

s

, and

false

otherwise.

Example 1:

Input:

s = "anagram", t = "nagaram"

Output:

true

Example 2:

Input:

s = "rat", t = "car"

Output:

false

Constraints:

1 <= s.length, t.length <= 5 * 10

4

s

and

t

consist of lowercase English letters.

Follow up:

What if the inputs contain Unicode characters? How would you adapt your solution to such a case?

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

```
/**
* @param {string} s
```

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
```

```
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (is-anagram s t)
(-> string? string? boolean?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Valid Anagram
* Difficulty: Easy
* Tags: string, hash, sort
```

```
*
* 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 isAnagram(string s, string t) {

}
};
```

Java Solution:

```
/**
* Problem: Valid Anagram
* Difficulty: Easy
* Tags: string, hash, sort
*
* 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 isAnagram(String s, String t) {

}
}
```

Python3 Solution:

```
"""
Problem: Valid Anagram
Difficulty: Easy
Tags: string, hash, sort

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 isAnagram(self, s: str, t: str) -> bool:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Valid Anagram
* Difficulty: Easy
* Tags: string, hash, sort
*
* 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} t
* @return {boolean}
*/
var isAnagram = function(s, t) {

};
```

TypeScript Solution:

```
/**
* Problem: Valid Anagram
* Difficulty: Easy
* Tags: string, hash, sort
*
* 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 isAnagram(s: string, t: string): boolean {

};
```

C# Solution:

```
/*
* Problem: Valid Anagram
* Difficulty: Easy
* Tags: string, hash, sort
*
* 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 IsAnagram(string s, string t) {

}
}
```

C Solution:

```
/*
* Problem: Valid Anagram
* Difficulty: Easy
* Tags: string, hash, sort
*
* 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 isAnagram(char* s, char* t) {

}
```

Go Solution:

```
// Problem: Valid Anagram
// Difficulty: Easy
// Tags: string, hash, sort
//
// 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 isAnagram(s string, t string) bool {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Valid Anagram
// Difficulty: Easy
// Tags: string, hash, sort
```

```
//
// 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 is_anagram(s: String, t: String) -> bool {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (is-anagram s t)
(-> string? string? boolean?)
)
```