Problem 2828: Check if a String Is an Acronym of Words

Problem Information

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

Problem Description

Given an array of strings

words

and a string

s

, determine if

s

is an

acronym

of words.

The string

s

is considered an acronym of

words

if it can be formed by concatenating the

first

character of each string in

words

in order

. For example,

"ab"

can be formed from

["apple", "banana"]

, but it can't be formed from

["bear", "aardvark"]

.

Return

true

if

s

is an acronym of

words

, and

false

otherwise.

Example 1:

Input:

words = ["alice","bob","charlie"], s = "abc"

Output:

true

Explanation:

The first character in the words "alice", "bob", and "charlie" are 'a', 'b', and 'c', respectively. Hence, s = "abc" is the acronym.

Example 2:

Input:

words = ["an","apple"], s = "a"

Output:

false

Explanation:

The first character in the words "an" and "apple" are 'a' and 'a', respectively. The acronym formed by concatenating these characters is "aa". Hence, s = "a" is not the acronym.

Example 3:

Input:

words = ["never","gonna","give","up","on","you"], s = "ngguoy"

Output:

true

Explanation:

By concatenating the first character of the words in the array, we get the string "ngguoy". Hence, s = "ngguoy" is the acronym.

Constraints:

1 <= words.length <= 100

1 <= words[i].length <= 10

1 <= s.length <= 100

words[i]

and

s

consist of lowercase English letters.

Code Snippets

C++:

```
class Solution {
public:
bool isAcronym(vector<string>& words, string s) {

}
};
```

Java:

```
class Solution {
public boolean isAcronym(List<String> words, String s) {

}
}
```

Python3:

```
class Solution:
def isAcronym(self, words: List[str], s: str) -> bool:
```

Python:

```
class Solution(object):
def isAcronym(self, words, s):
"""
:type words: List[str]
:type s: str
:rtype: bool
"""
```

JavaScript:

```
/**
* @param {string[]} words
* @param {string} s
* @return {boolean}
*/
var isAcronym = function(words, s) {

};
```

TypeScript:

```
function isAcronym(words: string[], s: string): boolean {

};
```

C#:

```
public class Solution {
public bool IsAcronym(IList<string> words, string s) {
}
}
```

C:

```
bool isAcronym(char** words, int wordsSize, char* s) {

}
```

Go:

```
func isAcronym(words []string, s string) bool {

}
```

Kotlin:

```
class Solution {
fun isAcronym(words: List<String>, s: String): Boolean {

}
}
```

Swift:

```
class Solution {
func isAcronym(_ words: [String], _ s: String) -> Bool {

}
}
```

Rust:

```
impl Solution {
pub fn is_acronym(words: Vec<String>, s: String) -> bool {

}
}
```

Ruby:

```
# @param {String[]} words
# @param {String} s
# @return {Boolean}
def is_acronym(words, s)

end
```

PHP:

```
class Solution {

/**
* @param String[] $words
* @param String $s
* @return Boolean
*/
function isAcronym($words, $s) {

}
}
```

Dart:

```
class Solution {
bool isAcronym(List<String> words, String s) {

}
}
```

Scala:

```
object Solution {
def isAcronym(words: List[String], s: String): Boolean = {

}
}
```

Elixir:

```
defmodule Solution do
@spec is_acronym(words :: [String.t], s :: String.t) :: boolean
def is_acronym(words, s) do
```

```
end
end
```

Erlang:

```
-spec is_acronym(Words :: [unicode:unicode_binary()], S ::
unicode:unicode_binary()) -> boolean().
is_acronym(Words, S) ->
.
```

Racket:

```
(define/contract (is-acronym words s)
(-> (listof string?) string? boolean?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Check if a String Is an Acronym of Words
* Difficulty: Easy
* Tags: array, string
*
* 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 isAcronym(vector<string>& words, string s) {

}
};
```

Java Solution:

```
/**
* Problem: Check if a String Is an Acronym of Words
* Difficulty: Easy
* Tags: array, string
*
* 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 isAcronym(List<String> words, String s) {

}
}
```

Python3 Solution:

```
"""
Problem: Check if a String Is an Acronym of Words
Difficulty: Easy
Tags: array, string

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 isAcronym(self, words: List[str], s: str) -> bool:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def isAcronym(self, words, s):
"""
:type words: List[str]
:type s: str
:rtype: bool
"""
```

JavaScript Solution:

```javascript
/**
* Problem: Check if a String Is an Acronym of Words
* Difficulty: Easy
* Tags: array, string
*
* 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[]} words
* @param {string} s
* @return {boolean}
*/
var isAcronym = function(words, s) {

};
```

TypeScript Solution:

```typescript
/**
* Problem: Check if a String Is an Acronym of Words
* Difficulty: Easy
* Tags: array, string
*
* 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 isAcronym(words: string[], s: string): boolean {

};
```

C# Solution:

```csharp
/*
* Problem: Check if a String Is an Acronym of Words
* Difficulty: Easy
* Tags: array, string
*
* 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 IsAcronym(IList<string> words, string s) {

}
}
```

C Solution:

```
/*
* Problem: Check if a String Is an Acronym of Words
* Difficulty: Easy
* Tags: array, string
*
* 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 isAcronym(char** words, int wordsSize, char* s) {

}
```

Go Solution:

```
// Problem: Check if a String Is an Acronym of Words
// Difficulty: Easy
// Tags: array, string
//
// 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 isAcronym(words []string, s string) bool {

}
```

Kotlin Solution:

```
class Solution {
fun isAcronym(words: List<String>, s: String): Boolean {

}
}
```

Swift Solution:

```
class Solution {
func isAcronym(_ words: [String], _ s: String) -> Bool {

}
}
```

Rust Solution:

```
// Problem: Check if a String Is an Acronym of Words
// Difficulty: Easy
// Tags: array, string
//
// 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 is_acronym(words: Vec<String>, s: String) -> bool {

}
}
```

Ruby Solution:

```
# @param {String[]} words
# @param {String} s
# @return {Boolean}
def is_acronym(words, s)
```

```
end
```

PHP Solution:

```
class Solution {

/**
* @param String[] $words
* @param String $s
* @return Boolean
*/
function isAcronym($words, $s) {

}
}
```

Dart Solution:

```
class Solution {
bool isAcronym(List<String> words, String s) {

}
}
```

Scala Solution:

```
object Solution {
def isAcronym(words: List[String], s: String): Boolean = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec is_acronym(words :: [String.t], s :: String.t) :: boolean
def is_acronym(words, s) do

end
end
```

Erlang Solution:

```
-spec is_acronym(Words :: [unicode:unicode_binary()], S ::
unicode:unicode_binary()) -> boolean().
is_acronym(Words, S) ->
.
```

Racket Solution:

```
(define/contract (is-acronym words s)
(-> (listof string?) string? boolean?)
)
```