Problem 953: Verifying an Alien Dictionary

Problem Information

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

Problem Description

In an alien language, surprisingly, they also use English lowercase letters, but possibly in a different

order

. The

order

of the alphabet is some permutation of lowercase letters.

Given a sequence of

words

written in the alien language, and the

order

of the alphabet, return

true

if and only if the given

words

are sorted lexicographically in this alien language.

Example 1:

Input:

words = ["hello","leetcode"], order = "hlabcdefgijkmnopqrstuvwxyz"

Output:

true

Explanation:

As 'h' comes before 'l' in this language, then the sequence is sorted.

Example 2:

Input:

words = ["word","world","row"], order = "worldabcefghijkmnpqstuvxyz"

Output:

false

Explanation:

As 'd' comes after 'l' in this language, then words[0] > words[1], hence the sequence is unsorted.

Example 3:

Input:

words = ["apple","app"], order = "abcdefghijklmnopqrstuvwxyz"

Output:

false

Explanation:

The first three characters "app" match, and the second string is shorter (in size.) According to lexicographical rules "apple" > "app", because 'l' > '∅', where '∅' is defined as the blank character which is less than any other character (

More info

).

Constraints:

1 <= words.length <= 100

1 <= words[i].length <= 20

order.length == 26

All characters in

words[i]

and

order

are English lowercase letters.

Code Snippets

C++:

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

```
}
};
```

Java:

```
class Solution {
public boolean isAlienSorted(String[] words, String order) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function isAlienSorted(words: string[], order: string): boolean {

};
```

C#:

```
public class Solution {
public bool IsAlienSorted(string[] words, string order) {

}
}
```

C:

```
bool isAlienSorted(char** words, int wordsSize, char* order) {

}
```

Go:

```
func isAlienSorted(words []string, order string) bool {

}
```

Kotlin:

```
class Solution {
fun isAlienSorted(words: Array<String>, order: String): Boolean {

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

```
class Solution {
bool isAlienSorted(List<String> words, String order) {

}
}
```

Scala:

```
object Solution {
def isAlienSorted(words: Array[String], order: String): Boolean = {

}
}
```

Elixir:

```
defmodule Solution do
@spec is_alien_sorted(words :: [String.t], order :: String.t) :: boolean
```

```
def is_alien_sorted(words, order) do

end
end
```

Erlang:

```
-spec is_alien_sorted(Words :: [unicode:unicode_binary()], Order ::
unicode:unicode_binary()) -> boolean().
is_alien_sorted(Words, Order) ->
.
```

Racket:

```
(define/contract (is-alien-sorted words order)
(-> (listof string?) string? boolean?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Verifying an Alien Dictionary
* Difficulty: Easy
* Tags: array, string, graph, hash, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

class Solution {
public:
bool isAlienSorted(vector<string>& words, string order) {

}
};
```

Java Solution:

```
/**
* Problem: Verifying an Alien Dictionary
* Difficulty: Easy
* Tags: array, string, graph, hash, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

class Solution {
public boolean isAlienSorted(String[] words, String order) {

}
}
```

Python3 Solution:

```
"""
Problem: Verifying an Alien Dictionary
Difficulty: Easy
Tags: array, string, graph, hash, sort

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) for hash map
"""

class Solution:
def isAlienSorted(self, words: List[str], order: str) -> bool:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Verifying an Alien Dictionary
* Difficulty: Easy
* Tags: array, string, graph, hash, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

/**
* @param {string[]} words
* @param {string} order
* @return {boolean}
*/
var isAlienSorted = function(words, order) {

};
```

TypeScript Solution:

```
/**
* Problem: Verifying an Alien Dictionary
* Difficulty: Easy
* Tags: array, string, graph, hash, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

function isAlienSorted(words: string[], order: string): boolean {

};
```

C# Solution:

```
/*
* Problem: Verifying an Alien Dictionary
* Difficulty: Easy
```

```
* Tags: array, string, graph, hash, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

public class Solution {
public bool IsAlienSorted(string[] words, string order) {

}
}
```

C Solution:

```
/*
* Problem: Verifying an Alien Dictionary
* Difficulty: Easy
* Tags: array, string, graph, hash, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

bool isAlienSorted(char** words, int wordsSize, char* order) {

}
```

Go Solution:

```
// Problem: Verifying an Alien Dictionary
// Difficulty: Easy
// Tags: array, string, graph, hash, sort
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

func isAlienSorted(words []string, order string) bool {
```

```
}
```

Kotlin Solution:

```
class Solution {
fun isAlienSorted(words: Array<String>, order: String): Boolean {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Verifying an Alien Dictionary
// Difficulty: Easy
// Tags: array, string, graph, hash, sort
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

impl Solution {
pub fn is_alien_sorted(words: Vec<String>, order: String) -> bool {

}
}
```

Ruby Solution:

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

```
end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

```
class Solution {
bool isAlienSorted(List<String> words, String order) {

}
}
```

Scala Solution:

```
object Solution {
def isAlienSorted(words: Array[String], order: String): Boolean = {

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

```
-spec is_alien_sorted(Words :: [unicode:unicode_binary()], Order ::
unicode:unicode_binary()) -> boolean().
is_alien_sorted(Words, Order) ->
.
```

Racket Solution:

```
(define/contract (is-alien-sorted words order)
(-> (listof string?) string? boolean?)
)
```