Problem 269: Alien Dictionary

Problem Information

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

Problem Description

There is a new alien language that uses the English alphabet. However, the order of the letters is unknown to you.

You are given a list of strings

words

from the alien language's dictionary. Now it is claimed that the strings in

words

are

sorted lexicographically

by the rules of this new language.

If this claim is incorrect, and the given arrangement of string in

words

cannot correspond to any order of letters, return

"".

Otherwise, return

a string of the unique letters in the new alien language sorted in

lexicographically increasing order

by the new language's rules

.

If there are multiple solutions, return

any of them

.

Example 1:

Input:

words = ["wrt","wrf","er","ett","rftt"]

Output:

"wertf"

Example 2:

Input:

words = ["z","x"]

Output:

"zx"

Example 3:

Input:

words = ["z","x","z"]

Output:

""

Explanation:

The order is invalid, so return

""

.

Constraints:

1 <= words.length <= 100

1 <= words[i].length <= 100

words[i]

consists of only lowercase English letters.

Code Snippets

C++:

```
class Solution {
public:
string alienOrder(vector<string>& words) {

}
};
```

Java:

```
class Solution {
public String alienOrder(String[] words) {
```

```
}
}
```

Python3:

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

Python:

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

JavaScript:

```
/**
* @param {string[]} words
* @return {string}
*/
var alienOrder = function(words) {

};
```

TypeScript:

```
function alienOrder(words: string[]): string {

};
```

C#:

```
public class Solution {
public string AlienOrder(string[] words) {

}
}
```

C:

```
char* alienOrder(char** words, int wordsSize) {

}
```

Go:

```
func alienOrder(words []string) string {

}
```

Kotlin:

```
class Solution {
fun alienOrder(words: Array<String>): String {

}
}
```

Swift:

```
class Solution {
func alienOrder(_ words: [String]) -> String {

}
}
```

Rust:

```
impl Solution {
pub fn alien_order(words: Vec<String>) -> String {

}
}
```

Ruby:

```
# @param {String[]} words
# @return {String}
def alien_order(words)

end
```

PHP:

```
class Solution {

/**
* @param String[] $words
* @return String
*/
function alienOrder($words) {

}
}
```

Dart:

```
class Solution {
String alienOrder(List<String> words) {

}
}
```

Scala:

```
object Solution {
def alienOrder(words: Array[String]): String = {

}
}
```

Elixir:

```
defmodule Solution do
@spec alien_order(words :: [String.t]) :: String.t
def alien_order(words) do

end
end
```

Erlang:

```
-spec alien_order(Words :: [unicode:unicode_binary()]) ->
unicode:unicode_binary().
alien_order(Words) ->
```

```
.
```

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: Alien Dictionary
* Difficulty: Hard
* Tags: array, string, graph, sort, search
*
* 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:
string alienOrder(vector<string>& words) {

}
};
```

Java Solution:

```
/**
* Problem: Alien Dictionary
* Difficulty: Hard
* Tags: array, string, graph, sort, search
*
* 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 String alienOrder(String[] words) {

}
}
```

Python3 Solution:

```
"""
Problem: Alien Dictionary
Difficulty: Hard
Tags: array, string, graph, sort, search

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Alien Dictionary
* Difficulty: Hard
* Tags: array, string, graph, sort, search
*
* 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
* @return {string}
*/
var alienOrder = function(words) {

};
```

TypeScript Solution:

```
/**
* Problem: Alien Dictionary
* Difficulty: Hard
* Tags: array, string, graph, sort, search
*
* 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 alienOrder(words: string[]): string {

};
```

C# Solution:

```
/*
* Problem: Alien Dictionary
* Difficulty: Hard
* Tags: array, string, graph, sort, search
*
* 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 string AlienOrder(string[] words) {

}
}
```

C Solution:

```
/*
* Problem: Alien Dictionary
* Difficulty: Hard
* Tags: array, string, graph, sort, search
*
* 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
*/

char* alienOrder(char** words, int wordsSize) {

}
```

Go Solution:

```
// Problem: Alien Dictionary
// Difficulty: Hard
// Tags: array, string, graph, sort, search
//
// 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 alienOrder(words []string) string {

}
```

Kotlin Solution:

```
class Solution {
fun alienOrder(words: Array<String>): String {

}
```

```
}
```

Swift Solution:

```
class Solution {
func alienOrder(_ words: [String]) -> String {

}
}
```

Rust Solution:

```
// Problem: Alien Dictionary
// Difficulty: Hard
// Tags: array, string, graph, sort, search
//
// 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 alien_order(words: Vec<String>) -> String {

}
}
```

Ruby Solution:

```
# @param {String[]} words
# @return {String}
def alien_order(words)

end
```

PHP Solution:

```
class Solution {

/**
* @param String[] $words
* @return String
```

```
*/
function alienOrder($words) {

}
}
```

Dart Solution:

```
class Solution {
String alienOrder(List<String> words) {

}
}
```

Scala Solution:

```
object Solution {
def alienOrder(words: Array[String]): String = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec alien_order(words :: [String.t]) :: String.t
def alien_order(words) do

end
end
```

Erlang Solution:

```
-spec alien_order(Words :: [unicode:unicode_binary()]) ->
unicode:unicode_binary().
alien_order(Words) ->
.
```

Racket Solution:

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