Problem 425: Word Squares

Problem Information

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

Problem Description

Given an array of

unique

strings

words

, return

all the

word squares

you can build from

words

. The same word from

words

can be used

multiple times

. You can return the answer in

any order

.

A sequence of strings forms a valid

word square

if the

k

th

row and column read the same string, where

0 <= k < max(numRows, numColumns)

.

For example, the word sequence

["ball","area","lead","lady"]

forms a word square because each word reads the same both horizontally and vertically.

Example 1:

Input:

words = ["area","lead","wall","lady","ball"]

Output:

[["ball","area","lead","lady"],["wall","area","lead","lady"]]

Explanation:

The output consists of two word squares. The order of output does not matter (just the order of words in each word square matters).

Example 2:

Input:

words = ["abat","baba","atan","atal"]

Output:

[["baba","abat","baba","atal"],["baba","abat","baba","atan"]]

Explanation:

The output consists of two word squares. The order of output does not matter (just the order of words in each word square matters).

Constraints:

1 <= words.length <= 1000

1 <= words[i].length <= 4

All

words[i]

have the same length.

words[i]

consists of only lowercase English letters.

All

words[i]

are

unique

.

Code Snippets

C++:

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

}
};
```

Java:

```
class Solution {
public List<List<String>> wordSquares(String[] words) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

```
public class Solution {
public IList<IList<string>> WordSquares(string[] words) {

}
}
```

C:

```
/**
* Return an array of arrays of size *returnSize.
* The sizes of the arrays are returned as *returnColumnSizes array.
* Note: Both returned array and *columnSizes array must be malloced, assume
caller calls free().
*/
char*** wordSquares(char** words, int wordsSize, int* returnSize, int**
returnColumnSizes) {

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (word-squares words)
(-> (listof string?) (listof (listof string?)))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Word Squares
* Difficulty: Hard
* 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:
vector<vector<string>> wordSquares(vector<string>& words) {

}
};
```

Java Solution:

```
/**
* Problem: Word Squares
* Difficulty: Hard
* 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 List<List<String>> wordSquares(String[] words) {

}
}
```

Python3 Solution:

```
"""
Problem: Word Squares
Difficulty: Hard
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 wordSquares(self, words: List[str]) -> List[List[str]]:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Word Squares
* Difficulty: Hard
* 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
* @return {string[][]}
*/
var wordSquares = function(words) {

};
```

TypeScript Solution:

```
/**
* Problem: Word Squares
* Difficulty: Hard
* 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 wordSquares(words: string[]): string[][] {

};
```

C# Solution:

```
/*
* Problem: Word Squares
* Difficulty: Hard
* 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 IList<IList<string>> WordSquares(string[] words) {

}
}
```

C Solution:

```
/*
* Problem: Word Squares
* Difficulty: Hard
* 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
```

```
*/

/**
* Return an array of arrays of size *returnSize.
* The sizes of the arrays are returned as *returnColumnSizes array.
* Note: Both returned array and *columnSizes array must be malloced, assume
caller calls free().
*/
char*** wordSquares(char** words, int wordsSize, int* returnSize, int**
returnColumnSizes) {

}
```

Go Solution:

```
// Problem: Word Squares
// Difficulty: Hard
// 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 wordSquares(words []string) [][]string {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}

}
```

Rust Solution:

```
// Problem: Word Squares
// Difficulty: Hard
// 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 word_squares(words: Vec<String>) -> Vec<Vec<String>> {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (word-squares words)
(-> (listof string?) (listof (listof string?)))
)
```