Problem 422: Valid Word Square

Problem Information

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

Problem Description

Given an array of strings

words

, return

true

if it forms a valid

word square

.

A sequence of strings forms a valid

word square

if the

k

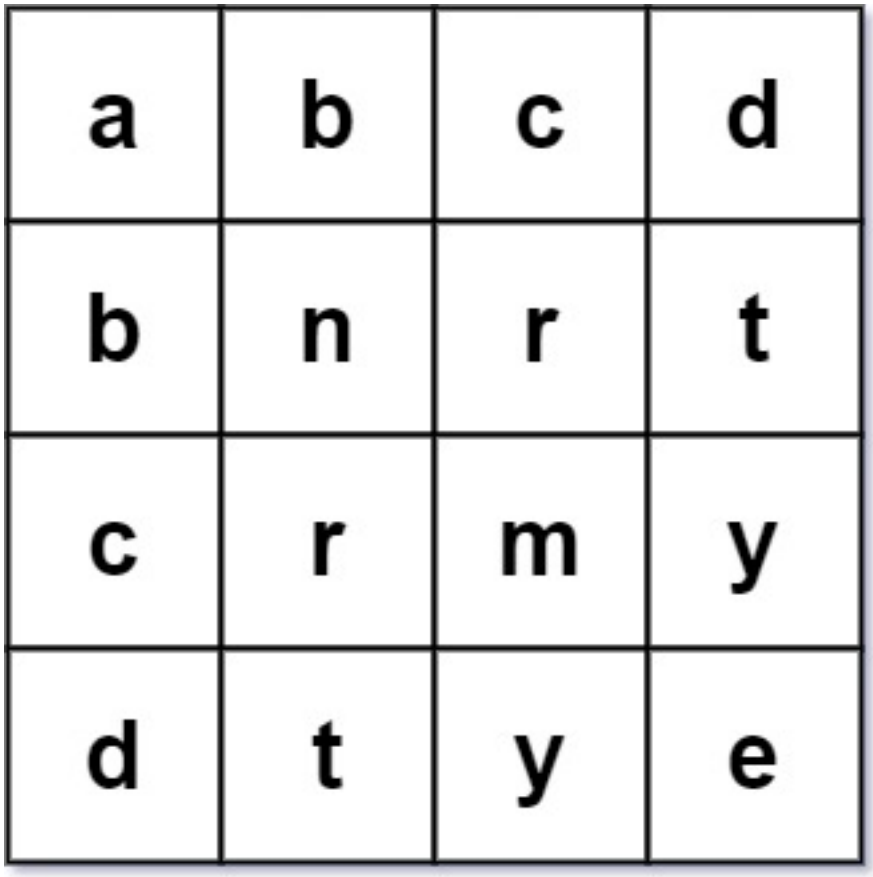
th

row and column read the same string, where

0 <= k < max(numRows, numColumns)

.

Example 1:



Input:

words = ["abcd","bnrt","crmy","dtye"]

Output:

true

Explanation:

The 1

st

row and 1

st

column both read "abcd". The 2

nd

row and 2

nd

column both read "bnrt". The 3

rd

row and 3

rd

column both read "crmy". The 4

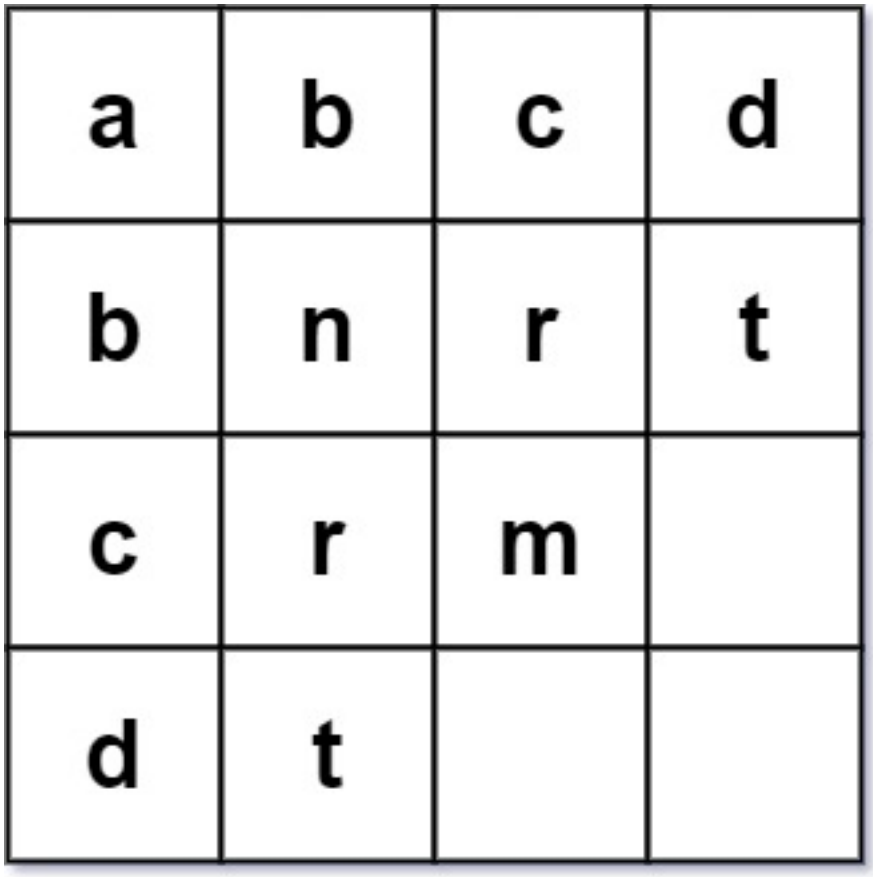
th

row and 4

th

column both read "dtye". Therefore, it is a valid word square.

Example 2:



Input:

words = ["abcd","bnrt","crm","dt"]

Output:

true

Explanation:

The 1

st

row and 1

st

column both read "abcd". The 2

nd

row and 2

nd

column both read "bnrt". The 3

rd

row and 3

rd

column both read "crm". The 4

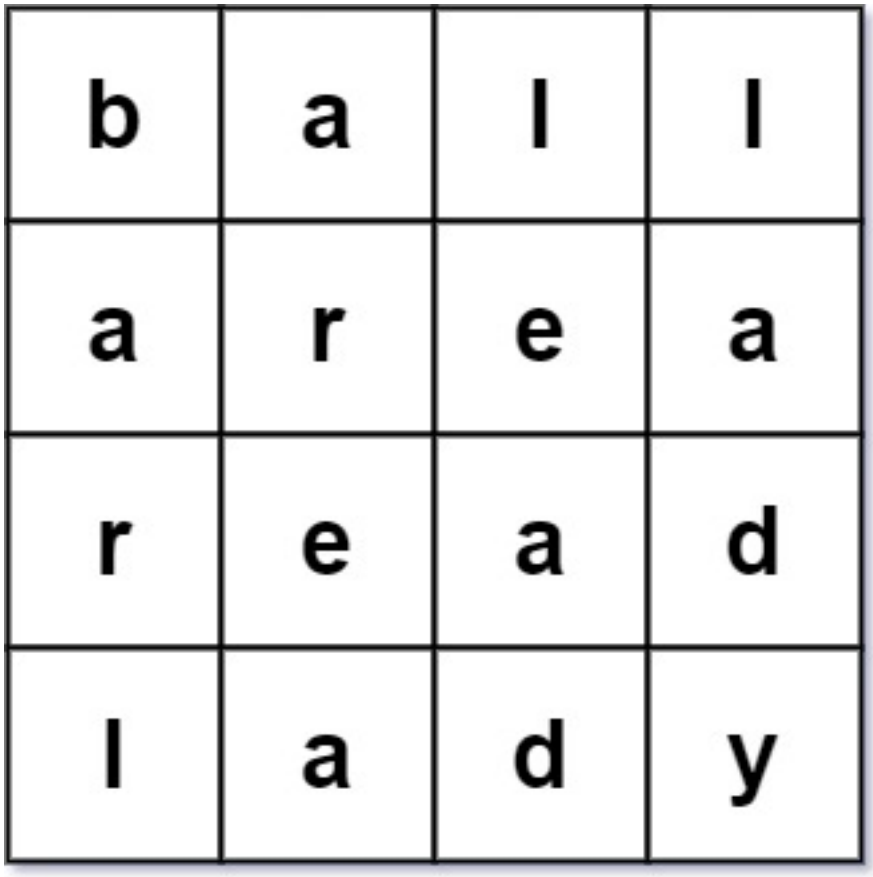
th

row and 4

th

column both read "dt". Therefore, it is a valid word square.

Example 3:



Input:

words = ["ball","area","read","lady"]

Output:

false

Explanation:

The 3

rd

row reads "read" while the 3

rd

column reads "lead". Therefore, it is NOT a valid word square.

Constraints:

1 <= words.length <= 500

1 <= words[i].length <= 500

words[i]

consists of only lowercase English letters.

Code Snippets

C++:

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

}
};
```

Java:

```
class Solution {
public boolean validWordSquare(List<String> words) {

}
}
```

Python3:

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

Python:

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

```
:rtype: bool
"""
```

JavaScript:

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

};
```

TypeScript:

```
function validWordSquare(words: string[]): boolean {

};
```

C#:

```
public class Solution {
public bool ValidWordSquare(IList<string> words) {

}
}
```

C:

```
bool validWordSquare(char** words, int wordsSize) {

}
```

Go:

```
func validWordSquare(words []string) bool {

}
```

Kotlin:

```
class Solution {
fun validWordSquare(words: List<String>): Boolean {

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

```
class Solution {
bool validWordSquare(List<String> words) {

}
}
```

Scala:

```
object Solution {
def validWordSquare(words: List[String]): Boolean = {

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (valid-word-square words)
(-> (listof string?) boolean?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Valid Word Square
* 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 validWordSquare(vector<string>& words) {

}
};
```

Java Solution:

```
/**
* Problem: Valid Word Square
* 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 validWordSquare(List<String> words) {

}
}
```

Python3 Solution:

```
"""
Problem: Valid Word Square
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 validWordSquare(self, words: List[str]) -> bool:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Valid Word Square
* 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
* @return {boolean}
*/
var validWordSquare = function(words) {

};
```

TypeScript Solution:

```
/**
* Problem: Valid Word Square
* 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 validWordSquare(words: string[]): boolean {

};
```

C# Solution:

```
/*
* Problem: Valid Word Square
* 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 ValidWordSquare(IList<string> words) {

}
}
```

C Solution:

```
/*
* Problem: Valid Word Square
* 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 validWordSquare(char** words, int wordsSize) {

}
```

Go Solution:

```
// Problem: Valid Word Square
// 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 validWordSquare(words []string) bool {

}
```

Kotlin Solution:

```
class Solution {
fun validWordSquare(words: List<String>): Boolean {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Valid Word Square
// 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 valid_word_square(words: Vec<String>) -> bool {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

```
class Solution {
bool validWordSquare(List<String> words) {

}
}
```

Scala Solution:

```
object Solution {
def validWordSquare(words: List[String]): Boolean = {

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (valid-word-square words)
(-> (listof string?) boolean?)
)
```