Problem 79: Word Search

Problem Information

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

Problem Description

Given an

m x n

grid of characters

board

and a string

word

, return

true

if

word

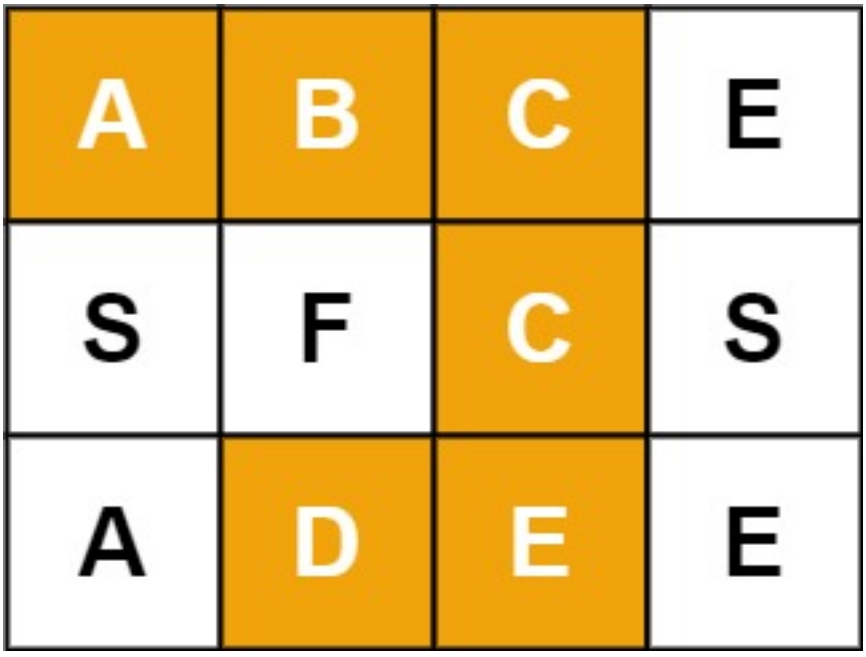
exists in the grid

.

The word can be constructed from letters of sequentially adjacent cells, where adjacent cells are horizontally or vertically neighboring. The same letter cell may not be used more than

once.

Example 1:



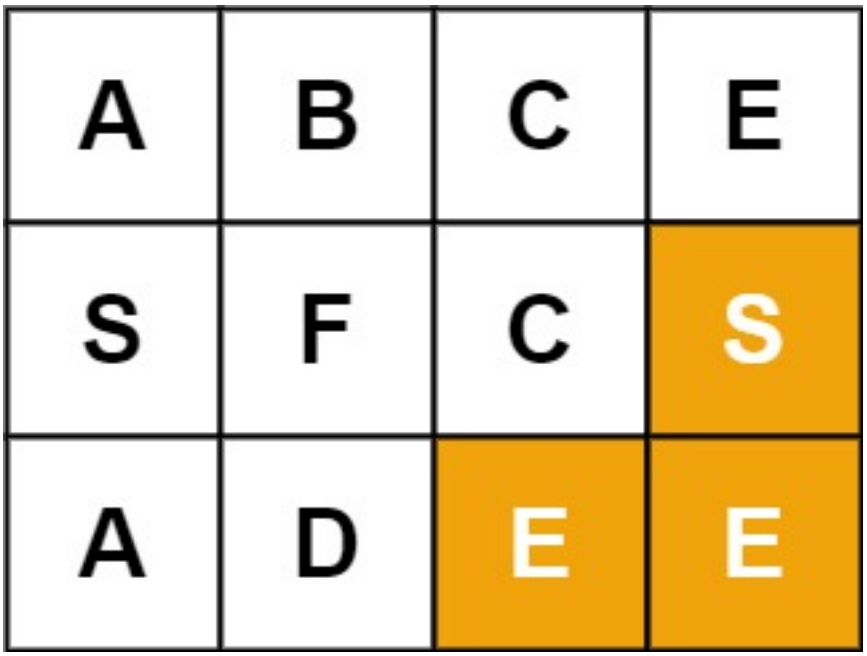
Input:

board = [["A","B","C","E"],["S","F","C","S"],["A","D","E","E"]], word = "ABCCED"

Output:

true

Example 2:



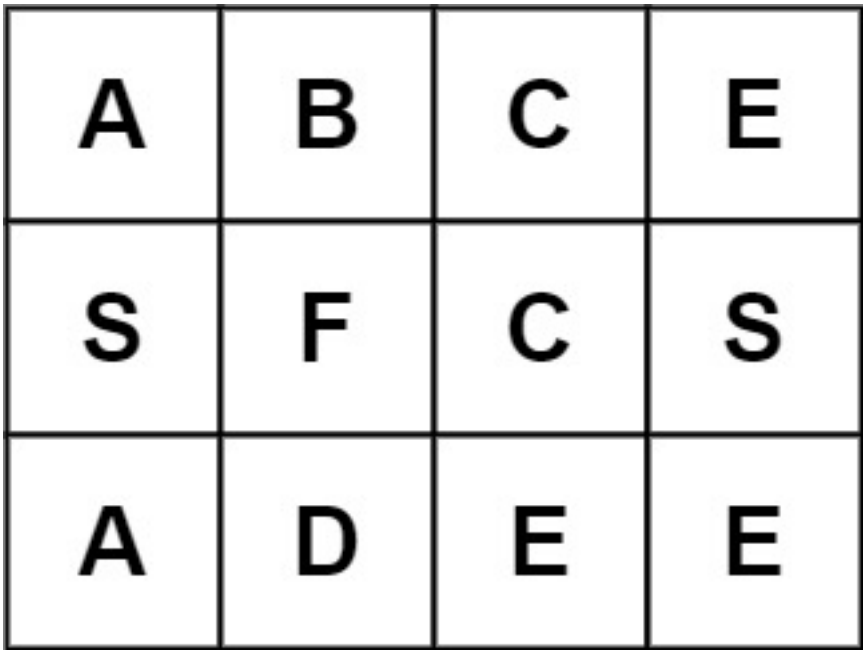
Input:

board = [["A","B","C","E"],["S","F","C","S"],["A","D","E","E"]], word = "SEE"

Output:

true

Example 3:



Input:

board = [["A","B","C","E"],["S","F","C","S"],["A","D","E","E"]], word = "ABCB"

Output:

false

Constraints:

m == board.length

n = board[i].length

1 <= m, n <= 6

1 <= word.length <= 15

board

and

word

consists of only lowercase and uppercase English letters.

Follow up:

Could you use search pruning to make your solution faster with a larger

board

?

Code Snippets

C++:

```
class Solution {
public:
bool exist(vector<vector<char>>& board, string word) {

}
};
```

Java:

```
class Solution {
public boolean exist(char[][] board, String word) {

}
}
```

Python3:

```
class Solution:
def exist(self, board: List[List[str]], word: str) -> bool:
```

Python:

```
class Solution(object):
def exist(self, board, word):
"""
:type board: List[List[str]]
:type word: str
:rtype: bool
"""
```

JavaScript:

```
/**
* @param {character[][]} board
* @param {string} word
* @return {boolean}
*/
var exist = function(board, word) {

};
```

TypeScript:

```
function exist(board: string[][], word: string): boolean {

};
```

C#:

```
public class Solution {
public bool Exist(char[][] board, string word) {

}
}
```

C:

```
bool exist(char** board, int boardSize, int* boardColSize, char* word) {

}
```

Go:

```
func exist(board [][]byte, word string) bool {

}
```

Kotlin:

```
class Solution {
fun exist(board: Array<CharArray>, word: String): Boolean {

}
}
```

Swift:

```
class Solution {
func exist(_ board: [[Character]], _ word: String) -> Bool {

}
}
```

Rust:

```
impl Solution {
pub fn exist(board: Vec<Vec<char>>, word: String) -> bool {

}
}
```

Ruby:

```
# @param {Character[][]} board
# @param {String} word
# @return {Boolean}
def exist(board, word)

end
```

PHP:

```
class Solution {

/**
* @param String[][] $board
* @param String $word
* @return Boolean
*/
function exist($board, $word) {

}
}
```

Dart:

```
class Solution {
bool exist(List<List<String>> board, String word) {

}
}
```

Scala:

```
object Solution {
def exist(board: Array[Array[Char]], word: String): Boolean = {

}
```

```
}
```

Elixir:

```
defmodule Solution do
@spec exist(board :: [[char]], word :: String.t) :: boolean
def exist(board, word) do

end
end
```

Erlang:

```
-spec exist(Board :: [[char()]], Word :: unicode:unicode_binary()) ->
boolean().
exist(Board, Word) ->
.
```

Racket:

```
(define/contract (exist board word)
(-> (listof (listof char?)) string? boolean?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Word Search
* Difficulty: Medium
* Tags: array, string, 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:
```

```
bool exist(vector<vector<char>>& board, string word) {

}
};
```

Java Solution:

```
/**
* Problem: Word Search
* Difficulty: Medium
* Tags: array, string, 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 boolean exist(char[][] board, String word) {

}
}
```

Python3 Solution:

```
"""
Problem: Word Search
Difficulty: Medium
Tags: array, string, 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 exist(self, board: List[List[str]], word: str) -> bool:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def exist(self, board, word):
"""
:type board: List[List[str]]
:type word: str
:rtype: bool
"""
```

JavaScript Solution:

```
/**
* Problem: Word Search
* Difficulty: Medium
* Tags: array, string, 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 {character[][]} board
* @param {string} word
* @return {boolean}
*/
var exist = function(board, word) {

};
```

TypeScript Solution:

```
/**
* Problem: Word Search
* Difficulty: Medium
* Tags: array, string, 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 exist(board: string[][], word: string): boolean {
```

```
};
```

C# Solution:

```
/*
* Problem: Word Search
* Difficulty: Medium
* Tags: array, string, 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 bool Exist(char[][] board, string word) {

}
}
```

C Solution:

```
/*
* Problem: Word Search
* Difficulty: Medium
* Tags: array, string, 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
*/

bool exist(char** board, int boardSize, int* boardColSize, char* word) {

}
```

Go Solution:

```
// Problem: Word Search
// Difficulty: Medium
```

```
// Tags: array, string, 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 exist(board [][]byte, word string) bool {

}
```

Kotlin Solution:

```
class Solution {
fun exist(board: Array<CharArray>, word: String): Boolean {

}
}
```

Swift Solution:

```
class Solution {
func exist(_ board: [[Character]], _ word: String) -> Bool {

}
}
```

Rust Solution:

```
// Problem: Word Search
// Difficulty: Medium
// Tags: array, string, 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 exist(board: Vec<Vec<char>>, word: String) -> bool {

}
}
```

Ruby Solution:

```
# @param {Character[][]} board
# @param {String} word
# @return {Boolean}
def exist(board, word)

end
```

PHP Solution:

```
class Solution {

/**
* @param String[][] $board
* @param String $word
* @return Boolean
*/
function exist($board, $word) {

}
}
```

Dart Solution:

```
class Solution {
bool exist(List<List<String>> board, String word) {

}
}
```

Scala Solution:

```
object Solution {
def exist(board: Array[Array[Char]], word: String): Boolean = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec exist(board :: [[char]], word :: String.t) :: boolean
def exist(board, word) do

end
end
```

Erlang Solution:

```
-spec exist(Board :: [[char()]], Word :: unicode:unicode_binary()) ->
boolean().
exist(Board, Word) ->
.
```

Racket Solution:

```
(define/contract (exist board word)
(-> (listof (listof char?)) string? boolean?)
)
```