Problem 212: Word Search II

Problem Information

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

Problem Description

Given an

m x n

board

of characters and a list of strings

words

, return

all words on the board

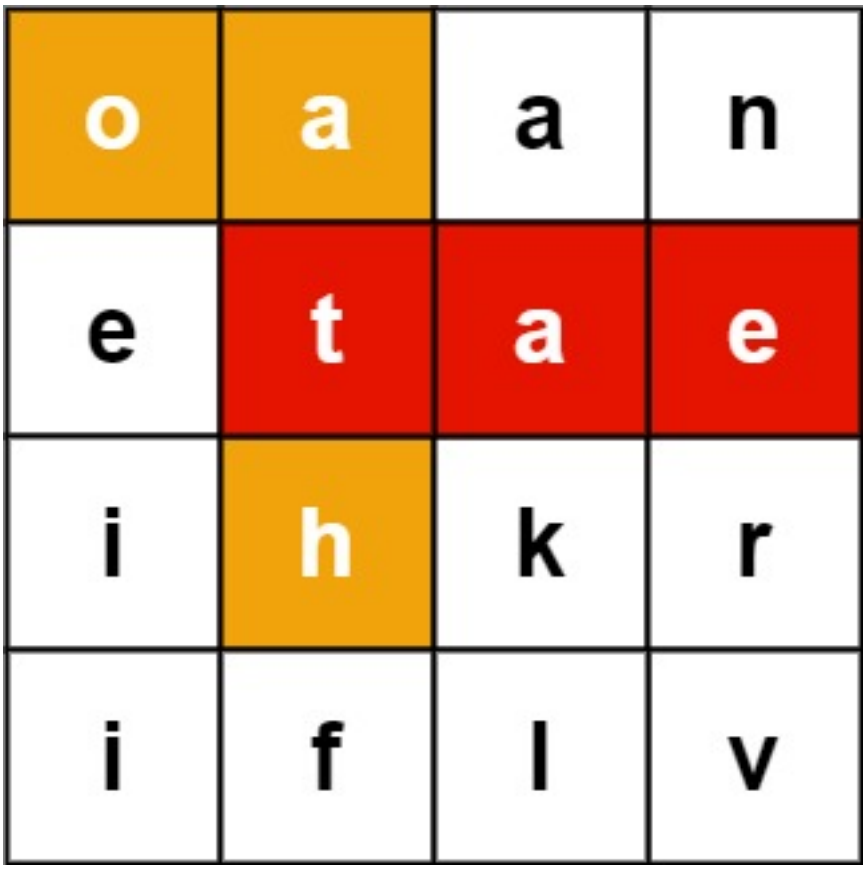
.

Each word must 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 in a word.

Example 1:



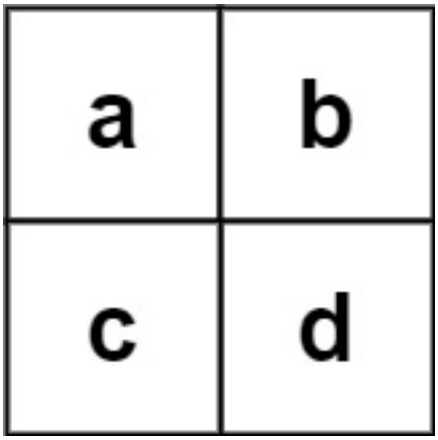
Input:

board = [["o","a","a","n"],["e","t","a","e"],["i","h","k","r"],["i","f","l","v"]], words = ["oath","pea","eat","rain"]

Output:

["eat","oath"]

Example 2:



Input:

board = [["a","b"],["c","d"]], words = ["abcb"]

Output:

[]

Constraints:

m == board.length

n == board[i].length

1 <= m, n <= 12

board[i][j]

is a lowercase English letter.

1 <= words.length <= 3 * 10

4

1 <= words[i].length <= 10

words[i]

consists of lowercase English letters.

All the strings of

words

are unique.

Code Snippets

C++:

```
class Solution {
public:
vector<string> findWords(vector<vector<char>>& board, vector<string>& words)
{

}
};
```

Java:

```
class Solution {
public List<String> findWords(char[][] board, String[] words) {

}
}
```

Python3:

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

Python:

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

JavaScript:

```
/**
* @param {character[][]} board
* @param {string[]} words
* @return {string[]}
*/
var findWords = function(board, words) {

};
```

TypeScript:

```
function findWords(board: string[][], words: string[]): string[] {

};
```

C#:

```
public class Solution {
public IList<string> FindWords(char[][] board, string[] words) {

}
}
```

C:

```
/**
* Note: The returned array must be malloced, assume caller calls free().
*/
char** findWords(char** board, int boardSize, int* boardColSize, char**
words, int wordsSize, int* returnSize) {

}
```

Go:

```
func findWords(board [][]byte, words []string) []string {

}
```

Kotlin:

```
class Solution {
fun findWords(board: Array<CharArray>, words: Array<String>): List<String> {

}
}
```

Swift:

```
class Solution {
func findWords(_ board: [[Character]], _ words: [String]) -> [String] {

}
}
```

Rust:

```
impl Solution {
pub fn find_words(board: Vec<Vec<char>>, words: Vec<String>) -> Vec<String> {

}
}
```

Ruby:

```
# @param {Character[][]} board
# @param {String[]} words
# @return {String[]}
def find_words(board, words)

end
```

PHP:

```
class Solution {

/**
* @param String[][] $board
* @param String[] $words
* @return String[]
*/
function findWords($board, $words) {

}
```

```
}
```

Dart:

```
class Solution {
List<String> findWords(List<List<String>> board, List<String> words) {

}
}
```

Scala:

```
object Solution {
def findWords(board: Array[Array[Char]], words: Array[String]): List[String]
= {

}
}
```

Elixir:

```
defmodule Solution do
@spec find_words(board :: [[char]], words :: [String.t]) :: [String.t]
def find_words(board, words) do

end
end
```

Erlang:

```
-spec find_words(Board :: [[char()]], Words :: [unicode:unicode_binary()]) ->
[unicode:unicode_binary()].
find_words(Board, Words) ->
.
```

Racket:

```
(define/contract (find-words board words)
(-> (listof (listof char?)) (listof string?) (listof string?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Word Search II
* Difficulty: Hard
* 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:
vector<string> findWords(vector<vector<char>>& board, vector<string>& words)
{

}
};
```

Java Solution:

```
/**
* Problem: Word Search II
* Difficulty: Hard
* 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 List<String> findWords(char[][] board, String[] words) {

}
}
```

Python3 Solution:

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Word Search II
* Difficulty: Hard
* 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[]} words
* @return {string[]}
*/
```

```
var findWords = function(board, words) {

};
```

TypeScript Solution:

```
/**
* Problem: Word Search II
* Difficulty: Hard
* 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 findWords(board: string[][], words: string[]): string[] {

};
```

C# Solution:

```
/*
* Problem: Word Search II
* Difficulty: Hard
* 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 IList<string> FindWords(char[][] board, string[] words) {

}
}
```

C Solution:

```
/*
* Problem: Word Search II
* Difficulty: Hard
* 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
*/

/**
* Note: The returned array must be malloced, assume caller calls free().
*/
char** findWords(char** board, int boardSize, int* boardColSize, char**
words, int wordsSize, int* returnSize) {

}
```

Go Solution:

```
// Problem: Word Search II
// Difficulty: Hard
// 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 findWords(board [][]byte, words []string) []string {

}
```

Kotlin Solution:

```
class Solution {
fun findWords(board: Array<CharArray>, words: Array<String>): List<String> {

}
}
```

Swift Solution:

```
class Solution {
func findWords(_ board: [[Character]], _ words: [String]) -> [String] {

}
}
```

Rust Solution:

```
// Problem: Word Search II
// Difficulty: Hard
// 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 find_words(board: Vec<Vec<char>>, words: Vec<String>) -> Vec<String> {

}
}
```

Ruby Solution:

```
# @param {Character[][]} board
# @param {String[]} words
# @return {String[]}
def find_words(board, words)

end
```

PHP Solution:

```
class Solution {

/**
* @param String[][] $board
* @param String[] $words
* @return String[]
*/
function findWords($board, $words) {
```

```
}
}
```

Dart Solution:

```
class Solution {
List<String> findWords(List<List<String>> board, List<String> words) {

}
}
```

Scala Solution:

```
object Solution {
def findWords(board: Array[Array[Char]], words: Array[String]): List[String]
= {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec find_words(board :: [[char]], words :: [String.t]) :: [String.t]
def find_words(board, words) do

end
end
```

Erlang Solution:

```
-spec find_words(Board :: [[char()]], Words :: [unicode:unicode_binary()]) ->
[unicode:unicode_binary()].
find_words(Board, Words) ->
.
```

Racket Solution:

```
(define/contract (find-words board words)
(-> (listof (listof char?)) (listof string?) (listof string?))
)
```