# Problem 2018: Check if Word Can Be Placed In Crossword

## Problem Information

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

## Problem Description

You are given an

m x n

matrix

board

, representing the

current

state of a crossword puzzle. The crossword contains lowercase English letters (from solved words),

' '

to represent any

empty

cells, and

'#'

to represent any

blocked

cells.

A word can be placed

horizontally

(left to right

or

right to left) or

vertically

(top to bottom

or

bottom to top) in the board if:

It does not occupy a cell containing the character

'#'

.

The cell each letter is placed in must either be

' '

(empty) or

match

the letter already on the

board

.

There must not be any empty cells

' '

or other lowercase letters

directly left or right

of the word if the word was placed

horizontally

.

There must not be any empty cells

' '

or other lowercase letters

directly above or below

the word if the word was placed

vertically

.

Given a string

word

, return

true

if

word

can be placed in

board

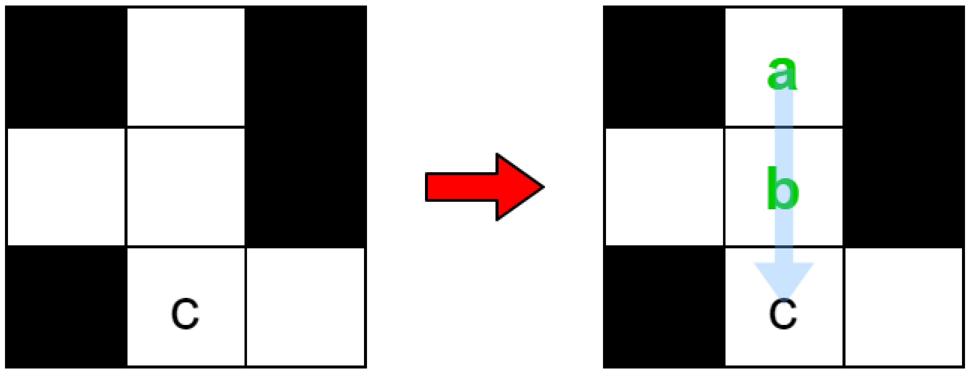
, or

false

otherwise

.

Example 1:



Input:

board = [["#", " ", "#"], [" ", " ", "#"], ["#", "c", " "]], word = "abc"

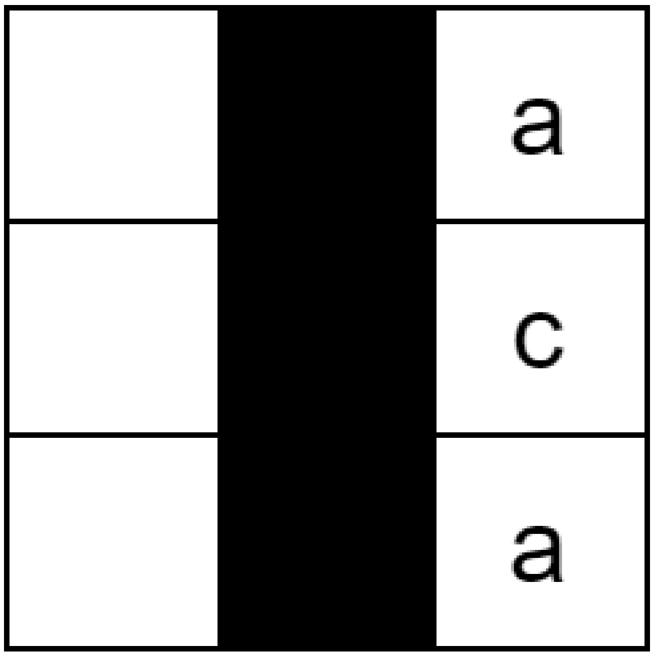
Output:

true

Explanation:

The word "abc" can be placed as shown above (top to bottom).

Example 2:



Input:

board = [[" ", "#", "a"], [" ", "#", "c"], [" ", "#", "a"]], word = "ac"

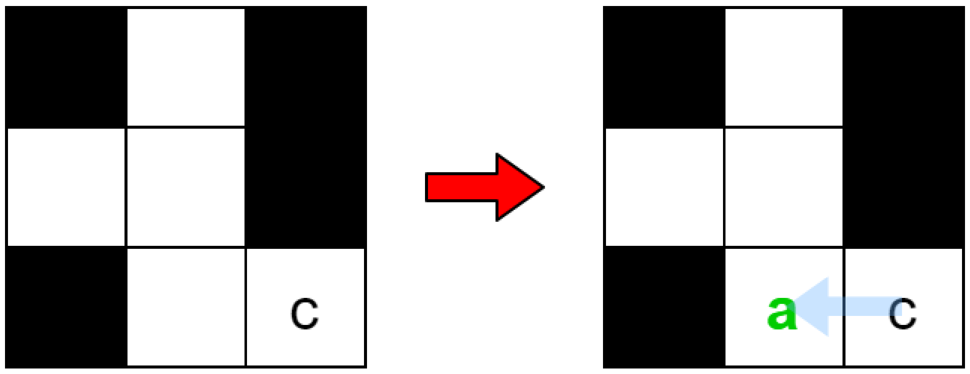
Output:

false

Explanation:

It is impossible to place the word because there will always be a space/letter above or below it.

Example 3:



Input:

board = [["#", " ", "#"], [" ", " ", "#"], ["#", " ", "c"]], word = "ca"

Output:

true

Explanation:

The word "ca" can be placed as shown above (right to left).

Constraints:

m == board.length

n == board[i].length

1 <= m * n <= 2 * 10

5

board[i][j]

will be

' '

,

'#'

, or a lowercase English letter.

1 <= word.length <= max(m, n)

word

will contain only lowercase English letters.

## Code Snippets

**C++:**

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

}
};
```

**Java:**

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

}
}
```

**Python3:**

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

**Python:**

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

```
:type word: str
:rtype: bool
"""
```

**JavaScript:**

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

};
```

**TypeScript:**

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

};
```

**C#:**

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

}
}
```

**C:**

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

}
```

**Go:**

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

}
```

**Kotlin:**

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

}
}
```

**Swift:**

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

}
}
```

**Rust:**

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

}
}
```

**Ruby:**

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

end
```

**PHP:**

```
class Solution {

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

**Dart:**

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

}
}
```

**Scala:**

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

}
}
```

**Elixir:**

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

end
end
```

**Erlang:**

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

**Racket:**

```
(define/contract (place-word-in-crossword board word)
(-> (listof (listof char?)) string? boolean?)
)
```

## Solutions

### C++ Solution:

```
/*
* Problem: Check if Word Can Be Placed In Crossword
* Difficulty: Medium
* 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 placeWordInCrossword(vector<vector<char>>& board, string word) {

}
};
```

### Java Solution:

```
/**
* Problem: Check if Word Can Be Placed In Crossword
* Difficulty: Medium
* 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 placeWordInCrossword(char[][] board, String word) {

}
}
```

### Python3 Solution:

```
"""
Problem: Check if Word Can Be Placed In Crossword
Difficulty: Medium
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 placeWordInCrossword(self, board: List[List[str]], word: str) -> bool:
# TODO: Implement optimized solution
pass
```

### Python Solution:

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

### JavaScript Solution:

```
/**
* Problem: Check if Word Can Be Placed In Crossword
* Difficulty: Medium
* 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 {character[][]} board
* @param {string} word
* @return {boolean}
*/
```

```
var placeWordInCrossword = function(board, word) {

};
```

### TypeScript Solution:

```
/**
* Problem: Check if Word Can Be Placed In Crossword
* Difficulty: Medium
* 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 placeWordInCrossword(board: string[][], word: string): boolean {

};
```

### C# Solution:

```
/*
* Problem: Check if Word Can Be Placed In Crossword
* Difficulty: Medium
* 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 PlaceWordInCrossword(char[][] board, string word) {

}
}
```

### C Solution:

```
/*
* Problem: Check if Word Can Be Placed In Crossword
* Difficulty: Medium
* 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 placeWordInCrossword(char** board, int boardSize, int* boardColSize,
char* word) {

}
```

### Go Solution:

```go
// Problem: Check if Word Can Be Placed In Crossword
// Difficulty: Medium
// 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 placeWordInCrossword(board [][]byte, word string) bool {

}
```

### Kotlin Solution:

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

}
}
```

### Swift Solution:

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

**Rust Solution:**

```
// Problem: Check if Word Can Be Placed In Crossword
// Difficulty: Medium
// 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 place_word_in_crossword(board: Vec<Vec<char>>, word: String) -> bool {

}
}
```

**Ruby Solution:**

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

end
```

**PHP Solution:**

```
class Solution {

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

}
}
```

### Dart Solution:

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

}
}
```

### Scala Solution:

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

}
}
```

### Elixir Solution:

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

end
end
```

### Erlang Solution:

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

### Racket Solution:

```
(define/contract (place-word-in-crossword board word)
(-> (listof (listof char?)) string? boolean?)
)
```