79. Word Search

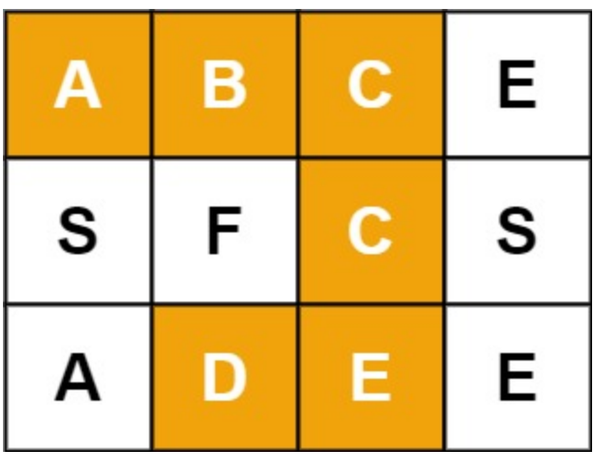
Difficulty	medium
☑	☑
Finished	@July 11, 2023
Problem	array
Previously asked company	Facebook
website	leetcode

Question:

Given an `m x n` grid of characters `board` and a string `word`, return `true` *if* `word` *exists in the grid*.

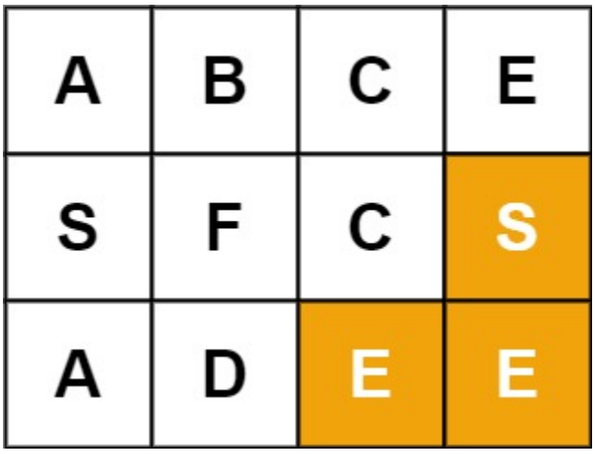
The word can be constructed from letters of sequentially adjacent cells, where adjacent cells are horizontally or vertically neighboring. The same letter cell may not be used more than once.

Example 1:



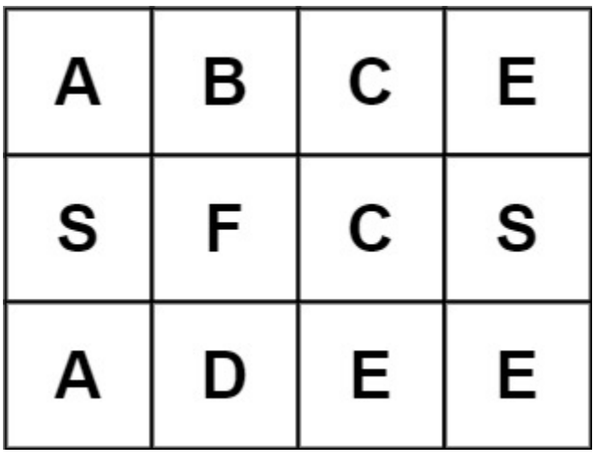

```
Input: board = [["A","B","C","E"],["S","F","C","S"],["A","D","E","E"]], word = "ABCCED"
Output: true
```

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
```

Optimal solution:

Time complexity: O(n*m*4^k) k = word length, n and m are matrix dimensions

Space complexity: O(1)

```
class Solution(object):
    def exist(self, board, word):
        ROWS, COLS = len(board), len(board[0])
        path = set()

        def dfs(r, c, i):
            if i == len(word):
                return True
            if (r < 0 or c < 0 or
                r >= ROWS or c >= COLS or
                word[i] != board[r][c] or (r, c) in path):
                return False
            path.add((r, c))
            res = (dfs(r-1,c,i+1) or
                   dfs(r+1,c,i+1) or
                   dfs(r,c-1,i+1) or
                   dfs(r,c+1,i+1))
```

```
            path.remove((r, c))
            return res

        for i in range(ROWS):
            for j in range(COLS):
                if dfs(i, j, 0): return True
```