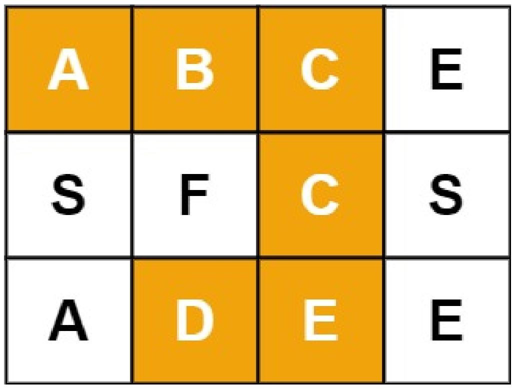
79. Word Search

Created	@July 10, 2021 6:03 AM
Difficulty	Medium
LC Url	https://leetcode.com/problems/word-search/
Importance	*****
Tag	Backtrack DFS
Video	https://www.youtube.com/watch?v=1zSg1WdmhIs

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

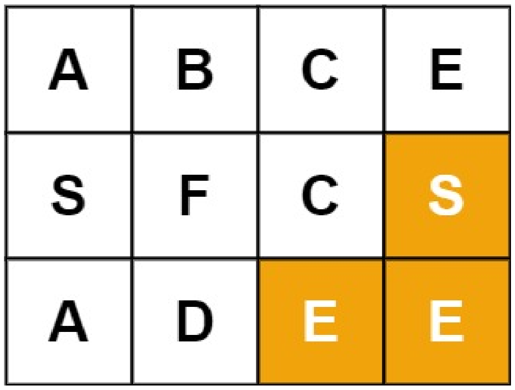
The word can be constructed from letters of sequentially adjacent cells, where adjacent cells are horizontally or vertically neighboring. The same letter cell may not be used more than once.

Example 1:



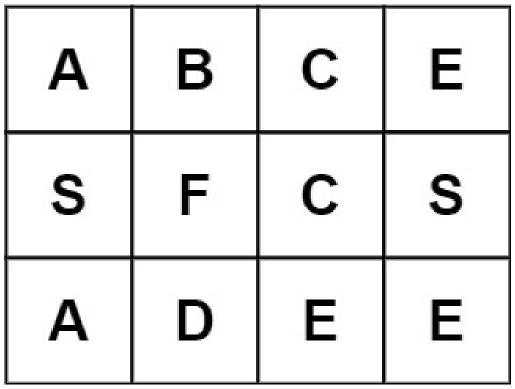
```
Input: board = [["A","B","C","E"],["S","F","C","S"],["A","D","E","E"]], word = "ABCCED"
Output: true
```

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.

Solution

整体思路

使用深度优先搜索（DFS）和回溯的思想实现。关于判断元素是否使用过，我用了一个二维数组 `mark` 对使用过的元素做标记。

外层：遍历

首先遍历 `board` 的所有元素，先找到和 `word` 第一个字母相同的元素，然后进入递归流程。假设这个元素的坐标为 `(i, j)`，进入递归流程前，先记得把该元素打上**使用过**的标记：

```
mark[i][j] = 1
```

内层：递归

好了，打完标记了，现在我们进入了递归流程。递归流程主要做了这么几件事：

1. 从 `(i, j)` 出发，朝它的上下左右试探，看看它周边的这四个元素是否能匹配 `word` 的下一个字母

- 如果匹配到了：带着该元素继续进入下一个递归
- 如果都匹配不到：返回 `False`

2. 当 `word` 的所有字母都完成匹配后，整个流程返回 `True`

几个注意点

- 递归时元素的坐标是否超过边界
- 回溯标记 `mark[i][j] = 0` 以及 `return` 的时机

```
class Solution:

    directions = [(0, 1), (0, -1), (1, 0), (-1, 0)]

    def exist(self, board: List[List[str]], word: str) -> bool:
        m = len(board)
        if m == 0:
            return False

        n = len(board[0])
        visited = [[0 for _ in range(n)] for _ in range(m)]

        for i in range(m):
            for j in range(n):
                if board[i][j] == word[0]:
                    visited[i][j] = 1
                    if self.backtrack(i, j, visited, board, word[1:]) == True:
                        return True
                    else:
                        visited[i][j] = 0
        return False

    def backtrack(self, i, j, visited, board, word):
        if len(word) == 0:
            return True

        for direct in self.directions:
            cur_i = i + direct[0]
            cur_j = j + direct[1]

            if 0 <= cur_i < len(board) and 0 <= cur_j < len(board[0]) and board[cur_i][cur_j] == word[0]:
                if visited[cur_i][cur_j] == 1:
                    continue
                visited[cur_i][cur_j] = 1
                if self.backtrack(cur_i, cur_j, visited, board, word[1:]) == True:
                    return True
                else:
                    visited[cur_i][cur_j] = 0
        return False
```

力扣

https://leetcode.cn/problems/word-search/solution/shen-du-you-xian-sou-suo-yu-hui-su-xiang-jie-by-ja/

```
class Solution:
    def exist(self, board: List[List[str]], word: str) -> bool:
        for i in range(len(board)):
            for j in range(len(board[0])):
                if self.dfs(board, i, j, word, 0):
                    return True
        return False

    def dfs(self, board, i, j, word, wordIndex):
        if wordIndex == len(word):
            return True

        if i < 0 or i >= len(board) or j < 0 or j >= len(board[0]) or word[wordIndex] != board[i][j]:
            return False

        temp = board[i][j]
        board[i][j] = ''

        found = self.dfs(board, i+1, j, word, wordIndex+1) \
                or self.dfs(board, i-1, j, word, wordIndex+1) \
                or self.dfs(board, i, j+1, word, wordIndex+1) \
                or self.dfs(board, i, j-1, word, wordIndex+1)

        board[i][j] = temp

        return found
```