

0079. 单词搜索

👤 [ITCharge](#) ⌚ 大约 2 分钟

- 标签：数组、回溯、矩阵
- 难度：中等

题目链接

- [0079. 单词搜索 - 力扣](#)

题目大意

描述： 给定一个 $m \times n$ 大小的二维字符矩阵 *board* 和一个字符串单词 *word*。

要求： 如果 *word* 存在于网格中，返回 `True`，否则返回 `False`。

说明：

- 单词必须按照字母顺序通过上下左右相邻的单元格字母构成。且同一个单元格内的字母不允许被重复使用。
- $m == board.length$ 。
- $n == board[i].length$ 。
- $1 \leq m, n \leq 6$ 。
- $1 \leq word.length \leq 15$ 。
- *board* 和 *word* 仅由大小写英文字母组成。

示例：

- 示例 1：

A	B	C	E
S	F	C	S
A	D	E	E

py

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

- 示例 2:

A	B	C	E
S	F	C	S
A	D	E	E

py

输入: board = `[["A","B","C","E"],["S","F","C","S"],["A","D","E","E"]]`, word = `"SEE"`
 输出: true

解题思路

思路 1：回溯算法

使用回溯算法在二维矩阵 *board* 中按照上下左右四个方向递归搜索。

设函数 `backtrack(i, j, index)` 表示从 `board[i][j]` 出发，能否搜索到单词字母 `word[index]`，以及 `index` 位置之后的后缀子串。如果能搜索到，则返回 `True`，否则返回 `False`。

`backtrack(i, j, index)` 执行步骤如下：

1. 如果 `board[i][j] = word[index]`，而且 `index` 已经到达 `word` 字符串末尾，则返回 `True`。
2. 如果 `board[i][j] = word[index]`，而且 `index` 未到达 `word` 字符串末尾，则遍历当前位置的所有相邻位置。如果从某个相邻位置能搜索到后缀子串，则返回 `True`，否则返回 `False`。
3. 如果 `board[i][j] ≠ word[index]`，则当前字符不匹配，返回 `False`。

思路 1：代码

```
class Solution:
    def exist(self, board: List[List[str]], word: str) -> bool:
        directs = [(0, 1), (0, -1), (1, 0), (-1, 0)]
        rows = len(board)
        if rows == 0:
            return False
        cols = len(board[0])
        visited = [[False for _ in range(cols)] for _ in range(rows)]

        def backtrack(i, j, index):
            if index == len(word) - 1:
                return board[i][j] == word[index]

            if board[i][j] == word[index]:
                visited[i][j] = True
                for direct in directs:
                    new_i = i + direct[0]
```

py

```
        new_j = j + direct[1]
        if 0 <= new_i < rows and 0 <= new_j < cols and
visited[new_i][new_j] == False:
            if backtrack(new_i, new_j, index + 1):
                return True
        visited[i][j] = False
    return False

for i in range(rows):
    for j in range(cols):
        if backtrack(i, j, 0):
            return True
return False
```

思路 1：复杂度分析

- **时间复杂度：** $O(m \times n \times 2^l)$ ，其中 m 、 n 为二维矩阵 *board* 的行数和列数。 l 为字符串 *word* 的长度。
- **空间复杂度：** $O(m \times n)$ 。