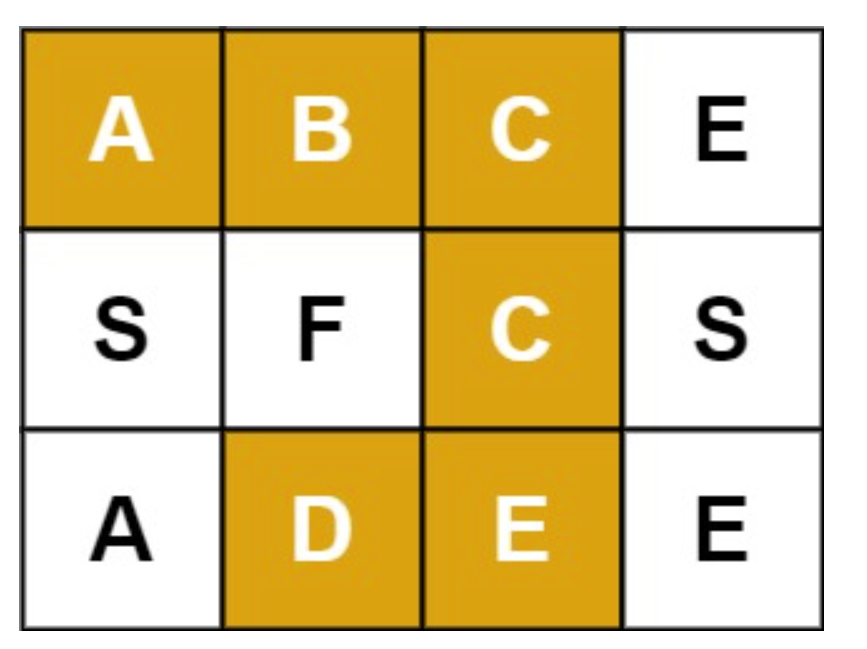
# 题目

给定一个 m x n 二维字符网格 board 和一个字符串单词 word 。如果 word 存在于网格中，返回 true ；否则，返回 false 。

单词必须按照字母顺序，通过相邻的单元格内的字母构成，其中“相邻”单元格是那些水平相邻或垂直相邻的单元格。同一个单元格内的字母不允许被重复使用。

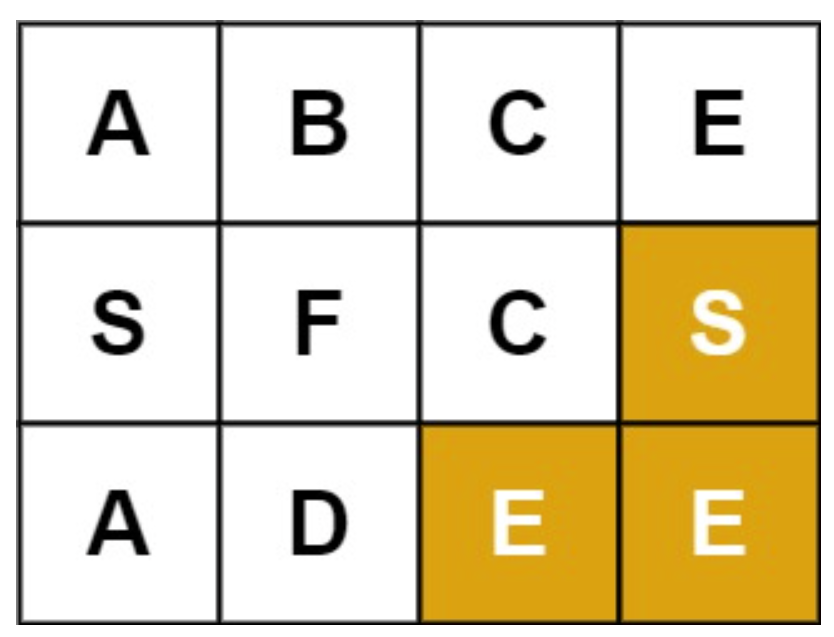
示例 1：



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

输出：true

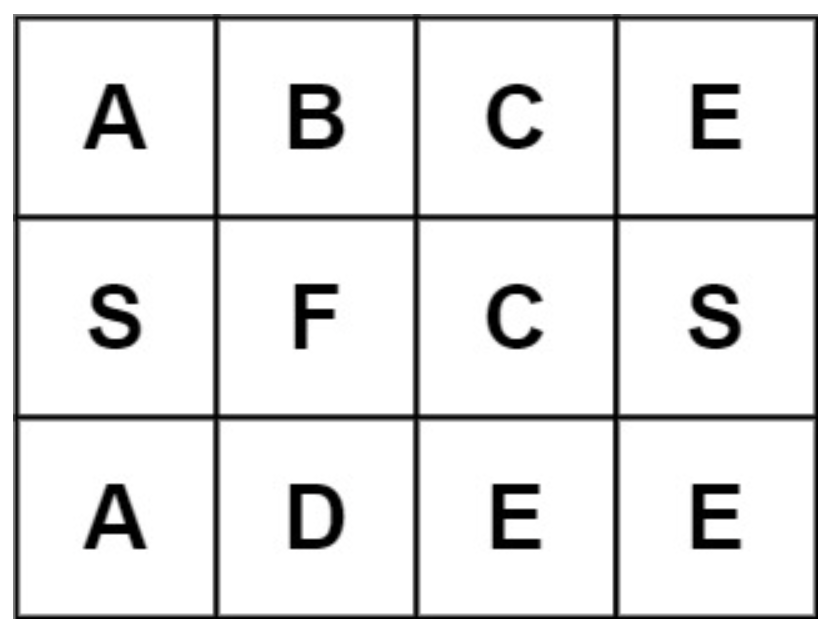
示例 2：



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

输出：true

示例 3：



输入：board = [["A","B","C","E"],["S","F","C","S"],["A","D","E","E"]], word = "ABCB"

输出：false

提示：

m == board.length

n = board[i].length

1 <= m, n <= 6

1 <= word.length <= 15

board 和 word 仅由大小写英文字母组成

进阶：你可以使用搜索剪枝的技术来优化解决方案，使其在 board 更大的情况下可以更快解决问题？

# 分析

## 方法一：回溯法

这个问题可以使用深度优先搜索（DFS）来解决。我们可以遍历二维字符网格中的每个位置，对于每个位置，如果该位置的字符与单词的第一个字符相同，则尝试从该位置开始匹配单词。

具体步骤如下：

1、遍历二维字符网格，对于每个位置 (i, j)，如果该位置的字符与单词的第一个字符相同，则从该位置开始尝试匹配单词。

2、在匹配过程中，使用递归来尝试匹配单词的每个字符。对于当前位置 (i, j)，如果其字符与单词的当前字符相同，则标记该位置已经访问，并尝试匹配单词的下一个字符。

3、如果成功匹配到单词的最后一个字符，则返回 true，否则返回 false。

代码：

class Solution {

public:

bool exist(vector<vector<char>>& board, string word) {

if (board.empty() || board[0].empty()) {

return false;

}

int m = board.size();

int n = board[0].size();

for (int i = 0; i < m; ++i) {

for (int j = 0; j < n; ++j) {

if (dfs(board, word, i, j, 0)) {

return true;

}

}

}

return false;

}

private:

bool dfs(vector<vector<char>>& board, string& word, int i, int j, int k) {

if (i < 0 || i >= board.size() || j < 0 || j >= board[0].size() || board[i][j] != word[k]) {

return false;

}

if (k == word.size() - 1) {

return true;

}

char tmp = board[i][j];

board[i][j] = '\0'; // 标记已访问

bool res = dfs(board, word, i + 1, j, k + 1) ||

dfs(board, word, i - 1, j, k + 1) ||

dfs(board, word, i, j + 1, k + 1) ||

dfs(board, word, i, j - 1, k + 1);

board[i][j] = tmp; // 恢复原字符

return res;

}

};

在这个实现中，我们定义了一个 `dfs` 函数来进行深度优先搜索，以尝试匹配单词。然后，在 `exist` 函数中，我们遍历二维字符网格中的每个位置，并从每个位置开始尝试匹配单词。如果成功匹配到单词的最后一个字符，则返回 true，否则返回 false。