Problem 79: Word Search

Problem Information

Difficulty: Medium
Acceptance Rate: 46.29%
Paid Only: No
Tags: Array, String, Backtracking, Depth-First Search, Matrix

Problem Description

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:

!\[\](https://assets.leetcode.com/uploads/2020/11/04/word2.jpg)

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

Example 2:

!\[\](https://assets.leetcode.com/uploads/2020/11/04/word-1.jpg)

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

Example 3:

!\[\](https://assets.leetcode.com/uploads/2020/10/15/word3.jpg)

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.

Follow up: Could you use search pruning to make your solution faster with a larger `board`?

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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