1. 题目
2. 题干

给定一个二维网格和一个单词，找出该单词是否存在于网格中。

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。

1. 示例

board =

[

['A','B','C','E'],

['S','F','C','S'],

['A','D','E','E']

]

给定 word = "ABCCED", 返回 true

给定 word = "SEE", 返回 true

给定 word = "ABCB", 返回 false

1. 题解
2. 思路

分析题干，这是一个迷宫似的寻找单词的问题，需要使用递归+回溯来解决。但是，这个递归好像很麻烦的样子。

首先，来看递归的结束条件，当然是找到的单词长度等于原始的单词长度了，我们可以通过一个不断增大的索引来判断。

其次，如何进入下一个递归呢？即如何在当前字母的周围找到下一个字母呢？这里还要考虑字母处于二维网格边界的问题，它可以通过判断坐标来解决；以及同一个单元格内的字母不能重复的问题，它可以通过创建一个标志数组来解决，如果某个坐标的字母已经被访问了，则置标志数组为true。如果可以进入下一个递归，且递归返回的是true，说明可以找到。

回溯即回到上一次递归的状态，即将标志数组的对应位置改为false即可。

只要对字母二维数组的每一个字母进行dfs判断即可，从cur=0开始，只要有返回true的，说明存在。

1. 代码实现

Java ：

class Solution {

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

        int m = board.length;

        int n = board[0].length;

        boolean[][] visited = new boolean[m][n];

        for(int i = 0; i < m; ++i){

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

                boolean flag = dfs(board, word, 0, visited , i, j);

                if(flag){

                    return true;

                }

            }

        }

        return false;

    }

    public boolean dfs(char[][]board,  String word,int cur, boolean[][] visited, int i, int j){

        if(board[i][j] != word.charAt(cur)){

            return false;

        }else{

            if( cur == word.length()-1){

                return true;

            }

        }

        visited[i][j] = true;

        int[][] directions = {{0,1}, {0, -1}, {1, 0}, {-1, 0}};

        boolean result = false;

        for(int[] direction: directions){

            int newi = i + direction[0], newj = j + direction[1];

            if(newi >= 0 && newi < board.length && newj >= 0 && newj < board[0].length){

                if(!visited[newi][newj]){

                    boolean flag = dfs(board, word, cur+1,visited, newi, newj);

                    if(flag){

                        result = true;

                        break;

                    }

                }

            }

        }

        visited[i][j] = false;

        return result;

    }

}