矩阵中的路径

请设计一个函数，用来判断在一个矩阵中是否存在一条包含某字符串所有字符的路径。路径可以从矩阵中的任意一格开始，每一步可以在矩阵中向左、右、上、下移动一格。如果一条路径经过了矩阵的某一格，那么该路径不能再次进入该格子。例如，在下面的 3×4 的矩阵中包含一条字符串“bfce”的路径（路径中的字母用加粗标出）。

[["a","b","c","e"],
["s","f","c","s"],
["a","d","e","e"]]

但矩阵中不包含字符串“abfb”的路径，因为字符串的第一个字符 b 占据了矩阵中的第一行第二个格子之后，路径不能再次进入这个格子。

示例 1：

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

输出：true

```
/*class Solution {
public:
    bool exist(vector<vector<char>>& board, string word) {
        if(word.empty()) return false;
        for(int i=0;i<board.size();i++)
        {
            for(int j=0;j<board[0].size();j++)
            {
                if(word[0]==board[i][j])
                {
                    if(dfs(board,word,i,j,0))
                        return true;
                }
            }
        }
        return false;
    }

    bool dfs(vector<vector<char>>& board, string& word,int i,int j,int w)
    {
        if(i<0||i>board.size()||j<0||j>board[0].size()||word[w]!=board[i][j
])
            return false;
        if(w==word.length()-1)
            return true;   //在矩阵中 找到字符串中最后一个字符
        char temp=board[i][j]; //记录所找到的当前字符，并在临时内存存储
        board[i][j]='\0';//将当前字符记录为'\0'，即一个不可能出现在 word 里的元
素，表明当前元素不可再参与比较， 放在矩阵中将不会再去查找比较
```

```
        if(dfs(board,word,i+1,j,w+1)||dfs(board,word,i-
1,j,w+1)||dfs(board,word,i,j+1,w+1)||dfs(board,word,i,j-1,w+1))
            return true;
        board[i][j]=temp;//将从临时内存放回矩阵空间  方便下一次回溯
        //如果以上条件不满足 证明在其上下左右 周围没有找到与 word 匹配的字符
         return false;
    }
};
*/

class Solution {
public:
    bool exist(vector<vector<char>>& board, string word) {
        if(word.empty()) return false;
        for(int i=0; i<board.size(); ++i)
        {
            for(int j=0; j<board[0].size(); ++j)
            {
                // 使用回溯法解题
                if(word[0]==board[i][j])
                {
                    if(dfs(board, word, i, j, 0)) return true;
                }
            }
        }
        return false;
    }
    bool dfs(vector<vector<char>>& board, string& word, int i, int j, int w
)
    {
        // 如果索引越界，或者值不匹配，返回 false
        if(i<0 || i>=board.size() || j<0 || j>=board[0].size() || board[i][
j]!=word[w]) return false;
        if(w == word.length() - 1) return true;
        char temp = board[i][j];
        board[i][j] = '\0'; // 将当前元素标记为'\0'，即一个不可能出现在 word 里
的元素，表明当前元素不可再参与比较
        if(dfs(board,word,i-1,j,w+1)
        || dfs(board,word,i+1,j,w+1)
        || dfs(board,word,i,j-1,w+1)
        || dfs(board,word,i,j+1,w+1))
        {
            // 当前元素的上下左右，如果有匹配到的，返回 true
            return true;
```

```
        }
        board[i][j] = temp; // 将当前元素恢复回其本身值
        return false;
    }
};
```