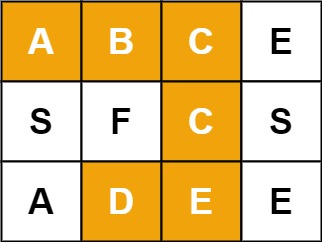
Given an m x n grid of characters board and a string word, return true *if* word *exists in the grid*.

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.

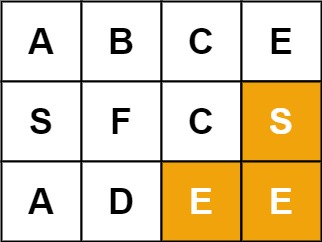
**Example 1:**



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

**Output:** true

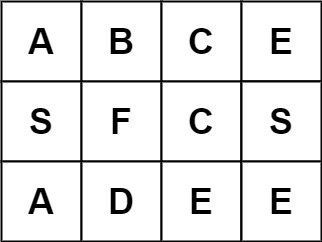
**Example 2:**



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

**Output:** true

**Example 3:**



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

**Output:** false

Solution:

class Solution {

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

for(int i=0; i<board.length; i++){

for(int j=0; j<board[i].length; j++){

if(board[i][j] == word.charAt(0) && dfs(board, i, j, 0, word))

return true;

}

}

return false;

}

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

if(count == word.length())

return true;

if(i<0 || i>=board.length || j<0 || j>= board[i].length || board[i][j] != word.charAt(count))

return false;

char temp = board[i][j];

board[i][j] = ' ';

boolean found = dfs(board, i+1, j, count+1, word)

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

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

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

board[i][j] = temp;

return found;

}

}