Word Search

Given a 2D board and a word, find if the word exists in the grid.

The word can be constructed from letters of sequentially adjacent cell, where "adjacent" cells are those horizontally or vertically neighboring. The same letter cell may not be used more than once.

For example,
Given **board** =

```
[
  ['A','B','C','E'],
  ['S','F','C','S'],
  ['A','D','E','E']
]
```

word = `"ABCCED"`, -> returns `true`,
word = `"SEE"`, -> returns `true`,
word = `"ABCB"`, -> returns `false`.

Solution 1

Here accepted solution based on recursion. To save memory I decuded to apply bit mask for every visited cell. Please check board[y][x] ^= 256;

```
public boolean exist(char[][] board, String word) {
    char[] w = word.toCharArray();
    for (int y=0; y<board.length; y++) {
        for (int x=0; x<board[y].length; x++) {
            if (exist(board, y, x, w, 0)) return true;
        }
    }
    return false;
}

private boolean exist(char[][] board, int y, int x, char[] word, int i) {
    if (i == word.length) return true;
    if (y<0 || x<0 || y == board.length || x == board[y].length) return false;
    if (board[y][x] != word[i]) return false;
    board[y][x] ^= 256;
    boolean exist = exist(board, y, x+1, word, i+1)
        || exist(board, y, x-1, word, i+1)
        || exist(board, y+1, x, word, i+1)
        || exist(board, y-1, x, word, i+1);
    board[y][x] ^= 256;
    return exist;
}
```

written by pavel-shlyk original link here

Solution 2

```
   class Solution {
    public:
         bool exist(vector<vector<char> > &board, string word) {
             m=board.size();
             n=board[0].size();
            for(int x=0;x<m;x++)
                for(int y=0;y<n;y++)
                {
                    if(isFound(board,word.c_str(),x,y))
                        return true;
                }
            return false;
        }
    private:
        int m;
        int n;
        bool isFound(vector<vector<char> > &board, const char* w, int x, int y)
        {
            if(x<0||y<0||x>=m||y>=n||board[x][y]=='\0'||*w!=board[x][y])
                return false;
            if(*(w+1)=='\0')
                return true;
            char t=board[x][y];
            board[x][y]='\0';
            if(isFound(board,w+1,x-1,y)||isFound(board,w+1,x+1,y)||isFound(board,
w+1,x,y-1)||isFound(board,w+1,x,y+1))
                return true;
            board[x][y]=t;
            return false;
        }
    };
```

written by pwh1 original link here

Solution 3

Typical dfs+backtracking question. It compare board[row][col] with word[start], if they match, change board[row][col] to '*' to mark it as visited. Then move to the next one (i.e. word[start+1]) and compare it to the current neighbors (doing it by recursion)

```
class Solution {
private:
    bool dfs(vector<vector<char>>& board, int row, int col, const string &word, i
nt start, int M, int N, int sLen)
    {
        char curC;
        bool res = false;
        if( (curC = board[row][col]) != word[start]) return false;
        if(start==sLen-1) return true;
        board[row][col] = '*';
        if(row>0) res = dfs(board, row-1, col, word, start+1, M, N, sLen);
        if(!res && row < M-1) res = dfs(board, row+1, col, word, start+1, M, N, s
Len);
        if(!res && col > 0)   res = dfs(board, row, col-1, word, start+1, M, N, s
Len);
        if(!res && col < N-1) res = dfs(board,  row, col+1, word, start+1, M, N,
sLen);
        board[row][col] = curC;
        return res;
    }

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

written by dong.wang.1694 original link here