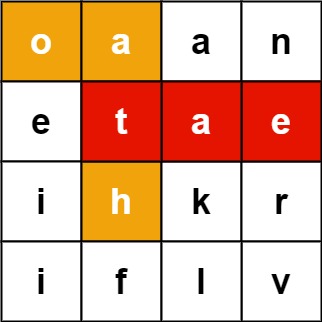
Given an m x n board of characters and a list of strings words, return *all words on the board*.

Each word must 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 in a word.

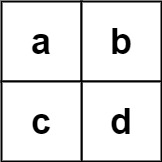
**Example 1:**



**Input:** board = [["o","a","a","n"],["e","t","a","e"],["i","h","k","r"],["i","f","l","v"]], words = ["oath","pea","eat","rain"]

**Output:** ["eat","oath"]

**Example 2:**



**Input:** board = [["a","b"],["c","d"]], words = ["abcb"]

**Output:** []

**Solution:**

class Solution {

public List<String> findWords(char[][] board, String[] words) {

List<String> ansList = new ArrayList<>();

if(board == null || board.length == 0)

return ansList;

TrieNode root = buildTrie(words);

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

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

dfs(board, i, j, root, ansList);

}

}

return ansList;

}

public void dfs(char[][] board, int i, int j, TrieNode p, List<String> res){

char c = board[i][j];

if (c == ' ' || p.children[c - 'a'] == null)

return;

p = p.children[c - 'a'];

if (p.word != null) { // found one

res.add(p.word);

p.word = null; // de-duplicate

}

board[i][j] = ' ';

if (i > 0) dfs(board, i - 1, j ,p, res);

if (j > 0) dfs(board, i, j - 1, p, res);

if (i < board.length - 1) dfs(board, i + 1, j, p, res);

if (j < board[0].length - 1) dfs(board, i, j + 1, p, res);

board[i][j] = c;

}

class TrieNode {

public String word;

public TrieNode[] children = new TrieNode[26];

public TrieNode() {}

}

public TrieNode buildTrie(String[] words) {

TrieNode root = new TrieNode();

for (String w : words) {

TrieNode ws = root;

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

char c = w.charAt(i);

if(ws.children[c - 'a'] == null){

ws.children[c - 'a'] = new TrieNode();

}

ws = ws.children[c - 'a'];

}

ws.word = w;

}

return root;

}

}