Lintcode. Word Search II

Given a matrix of lower alphabets and a dictionary. Find all words in the dictionary that can be found in the matrix. A word can start from any position in the matrix and go left/right/up/down to the adjacent position.

Given matrix:

doaf  
agai  
dcan

and dictionary:

{"dog", "dad", "dgdg", "can", "again"}

return {"dog", "dad", "can", "again"}

思路：BFS太慢。把dictionary建成Trie，在數組中找，可以減少重複搜索的次數。例如，dictionary中有doug 和 douglas。 用BFS需要在board中找doug前綴兩次。Trie不需重複搜索。doug前綴只搜了一次

public class Solution {

/\*\*

\* @param board: A list of lists of character

\* @param words: A list of string

\* @return: A list of string

\*/

public int []dx = {1, 0, -1, 0};

public int []dy = {0, 1, 0, -1};

public ArrayList<String> wordSearchII(char[][] board, ArrayList<String> words) {

// write your code here

ArrayList<String> ans = new ArrayList<String>();

if(board.length == 0)

return ans;

Trie tree = new Trie();

for(String word : words){

tree.insert(word);

}

boolean[][] visited = new boolean[board.length][board[0].length];

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

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

search(board, row, col, tree.root, visited, ans);

}

}

return ans;

}

private void search(char[][] board, int row, int col, TrieNode node, boolean[][] visited, ArrayList<String> ans){

if(node.isWord == true)

{

if(!ans.contains(node.val)){

ans.add(node.val);

}

}

if(row < 0 || row >= board.length || col < 0 || col >= board[0].length || visited[row][col] == true || node == null)

return ;

visited[row][col] = true;

if(node.map.containsKey(board[row][col])){

for(int i = 0; i < 4; i++){

char now = board[row][col];

search(board, row + dx[i], col + dy[i], node.map.get(now), visited, ans);

}

}

visited[row][col] = false;

}

}

class TrieNode{

String val = "";

HashMap<Character, TrieNode> map;

boolean isWord;

public TrieNode(){

map = new HashMap<Character, TrieNode>();

}

}

class Trie{

TrieNode root;

public Trie(){

root = new TrieNode();

}

public void insert(String str){

TrieNode curr = root;

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

char ch = str.charAt(i);

if(curr.map.get(ch) == null){

curr.map.put(ch, new TrieNode());

}

curr = curr.map.get(ch);

if(i == str.length() - 1){

curr.isWord = true;

curr.val = str;

}

}

}

public boolean find(String str){

TrieNode curr = root;

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

char ch = str.charAt(i);

if(curr.map.get(ch) == null)

return false;

curr = curr.map.get(ch);

}

return curr.isWord;

}

}