**Data structures and Algorithms Activity**

2024-2025: **ODD**

Section: **AF1**

**Course Instructor:** Dr.A. Jackulin Mahariba

**Couse Title:** 21CSC201J - Data Structures and Algorithms

*Submitted by*

**Name: Praneeth**

**[Reg. No.: RA2311026010300]**

**Name: Mahesh**

**[Reg. No.: RA2311026010290]**

**B.Tech. CSE – AIML**

**Problem Statemen**

**Question Description:**

**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.**

**Constraints:**

* **m == board.length**
* **n == board[i].length**
* **1 <= m, n <= 12**
* **board[i][j] is a lowercase English letter.**
* **1 <= words.length <= 3 \* 104**
* **1 <= words[i].length <= 10**
* **words[i] consists of lowercase English letters.**
* **All the strings of words are unique.**

**Algorithm For The Code:**

1. **Insert words into the Trie:**
   * **Initialize an empty TrieNode (root).**
   * **For each word in the words list, call the insert function:**
     + **Traverse each character of the word, creating a new TrieNode if needed for the corresponding character.**
     + **At the end of the word, store the pointer to the word (word) in the Trie node.**
2. **Start DFS for each board cell:**
   * **Traverse each cell (i, j) of the 2D grid (board).**
   * **For each cell, call dfs to explore all possible words starting from the cell if the character matches the starting letter of any word in the Trie.**
3. **DFS with backtracking:**
   * **The dfs function checks if the current character exists in the Trie.**
   * **If a word is found (word pointer is non-null), add the word to the result list (ans) and mark it as nullptr to avoid duplicate matches.**
   * **Recursively explore all four neighboring cells (up, down, left, right) while marking the current cell as visited by setting board[i][j] = '\*'.**
   * **After exploring all paths, restore the original character to allow other paths to use this cell again.**
4. **Return the result:**
   * **After DFS has been completed for all cells, return the list of found words (ans).**

**Program Code:**

**struct TrieNode {**

**vector<shared\_ptr<TrieNode>> children;**

**const string\* word = nullptr;**

**TrieNode() : children(26) {}**

**};**

**class Solution {**

**public:**

**vector<string> findWords(vector<vector<char>>& board, vector<string>& words) {**

**vector<string> ans;**

**for (const string& word : words)**

**insert(word);**

**for (int i = 0; i < board.size(); ++i)**

**for (int j = 0; j < board[0].size(); ++j)**

**dfs(board, i, j, root, ans);**

**return ans;**

**}**

**private:**

**shared\_ptr<TrieNode> root = make\_shared<TrieNode>();**

**void insert(const string& word) {**

**shared\_ptr<TrieNode> node = root;**

**for (const char c : word) {**

**const int i = c - 'a';**

**if (node->children[i] == nullptr)**

**node->children[i] = make\_shared<TrieNode>();**

**node = node->children[i];**

**}**

**node->word = &word;**

**}**

**void dfs(vector<vector<char>>& board, int i, int j, shared\_ptr<TrieNode> node,**

**vector<string>& ans) {**

**if (i < 0 || i == board.size() || j < 0 || j == board[0].size())**

**return;**

**if (board[i][j] == '\*')**

**return;**

**const char c = board[i][j];**

**shared\_ptr<TrieNode> child = node->children[c - 'a'];**

**if (child == nullptr)**

**return;**

**if (child->word != nullptr) {**

**ans.push\_back(\*child->word);**

**child->word = nullptr;**

**}**

**board[i][j] = '\*';**

**dfs(board, i + 1, j, child, ans);**

**dfs(board, i - 1, j, child, ans);**

**dfs(board, i, j + 1, child, ans);**

**dfs(board, i, j - 1, child, ans);**

**board[i][j] = c;**

**} };**

**SAMPLE INPUT AND OUPUT**

**Sample 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**

**["oath","eat"]**

**Expected**

**["oath","eat"]**

**Sample 2:**

**Input**

**board =**

**[["o","a","a","n"],["e","t","a","e"],["i","h","k","r"],["i","f","l","v"]]**

**words =**

**["oath","pea","eat","rain"]**

**Output**

**["oath","eat"]**

**Expected**

**["oath","eat"]**

**Result:**

**vector<vector<char>> board = {**

**{'o', 'a', 'a', 'n'},**

**{'e', 't', 'a', 'e'},**

**{'i', 'h', 'k', 'r'},**

**{'i', 'f', 'l', 'v'}**

**};**

**vector<string> words = {"oath", "pea", "eat", "rain"};**

**Final Output:**

**After performing DFS from all valid starting points on the board, the found words are:**

**["oath", "eat"]**

**Edge Cases Handled:**

1. **Empty Board:**
   * **If the board is empty, the output will be an empty list.**
2. **Empty Word List:**
   * **If the word list is empty, the output will be an empty list.**
3. **No Matching Words:**
   * **If none of the words in the word list can be formed from the characters in the board, the output will be an empty list.**
4. **Words Longer than the Board's Total Cells:**
   * **If a word is longer than the total number of cells in the board, it cannot be formed, and it won't appear in the result.**
5. **Single Cell Board:**
   * **If the board consists of a single cell, only single-character words matching that cell will be included in the output.**
6. **Duplicate Words in the Word List:**
   * **If there are duplicate words in the word list, only unique words will appear in the output.**
7. **Words Sharing Prefixes:**
   * **Words that share a common prefix will both be found if they exist in the board.**
8. **Backtracking:**
   * **The algorithm ensures that cells cannot be reused within the same word search, preventing invalid matches due to cell reuse.**

**Conclusion:**

**The code correctly handles various edge cases like empty inputs, no matches, single cell boards, words longer than the board, backtracking issues, and duplicate words. The Trie and DFS with backtracking provide an efficient way to solve the problem while ensuring correctness across these scenarios.**