

Word Search II

Trie + DFS on grid — prune entire branches ■■

● HARD

■ Hook

Given a board of characters and a list of words, find all words on the board. Naively: run DFS for each word separately. But with a Trie, **one DFS explores all words simultaneously** — pruning the search the moment no word can start with the current path.

■ Core Intuition — Simultaneous Search

■ The Trie-DFS marriage

Build a Trie from all words. Then DFS on the board, moving through the Trie simultaneously. If the current path is not a Trie prefix → stop (prune). If current node is a word end → add to results.

■ Visual

```
board:      words: ['oath', 'pea', 'eat', 'rain']
[o][a][a][n] Trie built from all 4 words.
[e][t][a][e]
[i][h][k][r] DFS from (0,0)='o':
[i][f][l][v] o → oa → oat → oath ✓ (found!)
              backtrack, try other directions...
              DFS from (1,0)='e':
              e → ea → eat ✓ (found!)
```

■ Code

```

class Solution {
    int[][] dirs = {{0,1},{0,-1},{1,0},{-1,0}};
    public List<String> findWords(char[][] board, String[] words) {
        TrieNode root = buildTrie(words);
        List<String> result = new ArrayList<>();
        int m=board.length, n=board[0].length;
        for (int i=0; i<m; i++)
            for (int j=0; j<n; j++)
                dfs(board, i, j, root, result);
        return result;
    }
    void dfs(char[][] board, int i, int j, TrieNode node, List<String> res) {
        char c = board[i][j];
        if (c=='#' || node.children[c-'a']==null) return; // pruned
        node = node.children[c-'a'];
        if (node.word != null) { res.add(node.word); node.word=null; } // found
        board[i][j] = '#'; // mark visited
        for (int[] d : dirs) {
            int ni=i+d[0], nj=j+d[1];
            if (ni>=0&&ni<board.length&&nj>=0&&nj<board[0].length)
                dfs(board, ni, nj, node, res);
        }
        board[i][j] = c; // restore
    }
    TrieNode buildTrie(String[] words) {
        TrieNode root = new TrieNode();
        for (String w : words) {
            TrieNode cur = root;
            for (char c : w.toCharArray()) {
                int i = c-'a';
                if (cur.children[i]==null) cur.children[i]=new TrieNode();
                cur = cur.children[i];
            }
            cur.word = w; // store word at leaf
        }
        return root;
    }
    static class TrieNode { TrieNode[] children=new TrieNode[26]; String word; }
}

```

■ TIME

$O(M \times N \times 4^L)$

■ SPACE

$O(W \times L)$

M,N=board dims, L=max word length, W=number of words. Trie pruning dramatically reduces constants.

- ✓ Build Trie from all words (store word string at terminal nodes)
- ✓ DFS from every board cell, traverse Trie simultaneously
- ✓ Prune if current char not in Trie children
- ✓ Mark visited with '#', restore after DFS
- ✓ Set node.word=null after finding to avoid duplicates

■ MENTAL MODEL — Remember This Forever

Without Trie: search for word1, then restart for word2, etc — $O(\text{words} \times \text{board} \times L)$. With Trie: one single DFS explores all words at once. The Trie tells you instantly if your current path is even a prefix of any word — pruning entire subtrees of the search space.