Word Search II

The trick to pass big inputs is: build a Trie for words, then DFS the board following trie edges, pruning dead paths early.

Plan:

1. Trie node:
   - TrieNode { TrieNode* next[26] {nullptr}; bool isEnd = false; string word; }
     - Store the full word at the terminal node (isEnd = true). That lets you push results in O(1) without carrying a path vector.
2. Build trie:
   - For each word:
     - walk / create children by idx = ch - 'a'
     - at the end: isEnd = true; node → word = word
3. DFS from every cell:
   - Arguments: (r, c, node)
   - If board[r][c] is not a child from node, stop.
   - Move to that child. If child→isEnd:
     - add child→isEnd = false (prevents duplicates if multiple paths reach it)
   - Mark the board cell as visited (e.g., set to '#'), explore 4 neighbors, restore after.
   - Prune: if after exploring you see child has no non-null children and !child→isEnd, you can delete that node (optional micro-optimization).
4. Visited handling:
   - Use in-place marking ('#') and restore the character when unwinding. Avoid a separate visited matrix for cache locality.

Boundaries & pruning:

- Bounds check (0 <= r < m, 0 <= c < n)
- If board[r][c] == '#' skip (already in path)
- Early cut: if current trie node has no child for board[r][c], return

immediately (huge pruning).

Complexity:

- Building trie: total word chars $\Sigma|w|$ ($\leq 3 \cdot 10^5$)
- DFS: Each cell explores at most 4 dirs, trie pruning kills most branches. This passes within limits.