AlgoExpert Quad Layout C++ 12px Sublime Monokai 00:00:00

Prompt Scratchpad Our Solution(s) Video Explanation Run Code

Solution 1

```
    trie.add(word);
  }
  unordered_map<string, bool> finalWords;
  vector<vector<bool>> visited(board.size(),
                               vector<bool>(board[0].size(), false));
  for (int i = 0; i < board.size(); i++) {
    for (int j = 0; j < board[0].size(); j++) {
      explore(i, j, board, trie.root, &visited, &finalWords);
    }
  }
  vector<string> finalWordsArray;
  for (auto it : finalWords) {
    finalWordsArray.push_back(it.first);
  }
  return finalWordsArray;
}

void explore(int i, int j, vector<vector<char>> board, TrieNode *trieNode,
             vector<vector<bool>> *visited,
             unordered_map<string, bool> *finalWords) {
  if (visited->at(i)[j]) {
    return;
  }
  char letter = board[i][j];
  if (trieNode->children.find(letter) == trieNode->children.end()) {
    return;
  }
  visited->at(i)[j] = true;
  trieNode = trieNode->children[letter];
  if (trieNode->children.find('*') != trieNode->children.end()) {
    finalWords->insert({trieNode->word, true});
  }
  vector<vector<int>> neighbors = getNeighbors(i, j, board);
  for (vector<int> neighbor : neighbors) {
    explore(neighbor[0], neighbor[1], board, trieNode, visited, finalWords);
  }
  visited->at(i)[j] = false;
}

vector<vector<int>> getNeighbors(int i, int j, vector<vector<char>> board) {
  vector<vector<int>> neighbors;
  if (i > 0 && j > 0) {
    neighbors.push_back({i - 1, j - 1});
  }
  if (i > 0 && j < board[0].size() - 1) {
    neighbors.push_back({i - 1, j + 1});
  }
  if (i < board.size() - 1 && j < board[0].size() - 1) {
    neighbors.push_back({i + 1, j + 1});
  }
  if (i < board.size() - 1 && j > 0) {
    neighbors.push_back({i + 1, j - 1});
  }
  if (i > 0) {
    neighbors.push_back({i - 1, j});
  }
  if (i < board.size() - 1) {
    neighbors.push_back({i + 1, j});
  }
  if (j > 0) {
    neighbors.push_back({i, j - 1});
  }
  if (j < board[0].size() - 1) {
    neighbors.push_back({i, j + 1});
  }
  return neighbors;
}

Trie::Trie() {
  this->root = new TrieNode();
  this->endSymbol = '*';
}

void Trie::add(string str) {
  TrieNode *node = this->root;
  for (char letter : str) {
    if (node->children.find(letter) == node->children.end()) {
      TrieNode *newNode = new TrieNode();
      node->children.insert({letter, newNode});
    }
    node = node->children[letter];
  }
  node->children.insert({this->endSymbol, NULL});
  node->word = str;
}

```