

## 1. Number of Islands

**200. Number of Islands** Solved

Medium Topics Companies

Given an  $m \times n$  2D binary grid `grid` which represents a map of '1's (land) and '0's (water), return the number of islands.

An island is surrounded by water and is formed by connecting adjacent lands horizontally or vertically. You may assume all four edges of the grid are all surrounded by water.

**Example 1:**

Input: `grid = [ [ "1","1","1","1","0"], [ "1","1","0","1","0"], [ "1","1","0","0","0"], [ "0","0","0","0","0"] ]`

Output: 1

**Example 2:**

Input: `grid = [ [ "1","1","0","0","0"], [ "1","1","0","0","0"], [ "0","0","0","0","0"], [ "0","0","0","1","1"] ]`

Output: 2

23.7K 251 553 Online

```
C++
q.push({r, c});

while (!q.empty()) {
    auto [row, col] = q.front();
    q.pop();

    for (auto [dr, dc] : directions) {
        int nr = row + dr;
        int nc = col + dc;
        if (nr >= 0 && nr < rows && nc >= 0 && nc < cols && grid[nr][nc] == '1' && !visited[nr][nc]) {
            q.push({nr, nc});
            visited[nr][nc] = true;
        }
    }
}
```

Accepted Runtime: 3 ms

Case 1 Case 2

Input: `grid = [ [ "1","1","1","1","1","1","1","1","1","1"], [ "1","1","1","1","1","1","1","1","1","1"], [ "1","1","1","1","1","1","1","1","1","1"], [ "1","1","1","1","1","1","1","1","1","1"], [ "1","1","1","1","1","1","1","1","1","1"], [ "1","1","1","1","1","1","1","1","1","1"], [ "1","1","1","1","1","1","1","1","1","1"], [ "1","1","1","1","1","1","1","1","1","1"], [ "1","1","1","1","1","1","1","1","1","1"], [ "1","1","1","1","1","1","1","1","1","1"] ]`

## 2. Word Ladder

**127. Word Ladder** Solved

Hard Topics Companies

A transformation sequence from word `beginWord` to word `endWord` using a dictionary `wordList` is a sequence of words `beginWord` → `s1` → `s2` → ... → `sk` such that:

- Every adjacent pair of words differs by a single letter.
- Every `si` for  $1 \leq i \leq k$  is in `wordList`. Note that `beginWord` does not need to be in `wordList`.
- `sk` == `endWord`.

Given two words, `beginWord` and `endWord`, and a dictionary `wordList`, return the number of words in the shortest transformation sequence from `beginWord` to `endWord`, or 0 if no such sequence exists.

**Example 1:**

Input: `beginWord = "hit", endWord = "cog", wordList = ["hot","dot","dog","lot","log","cog"]`

Output: 5

Explanation: One shortest transformation sequence is "hit" -> "hot" -> "dot" -> "dog" -> "cog", which is 5 words long.

**Example 2:**

Input: `beginWord = "hit", endWord = "cog", wordList = ["hot","dot","dog","lot","log"]`

Output: 0

12.6K 186 305 Online

```
C++
dict.erase(word);
for (int j = 0; j < word.size(); j++) {
    char c = word[j];
    for (int k = 0; k < 26; k++) {
        word[j] = 'a' + k;
        if (dict.find(word) != dict.end()) {
            todo.push(word);
        }
    }
    word[j] = c;
}

ladder++;
return 0;
}
```

Accepted Runtime: 0 ms

Case 1 Case 2

Input: `beginWord = "hit"`

### 3. Surrounded Regions

130. Surrounded Regions

Medium

You are given an  $m \times n$  matrix `board` containing letters 'X' and 'O', capture regions that are surrounded:

- Connect**: A cell is connected to adjacent cells horizontally or vertically.
- Region**: To form a region connect every 'O' cell.
- Surround**: The region is surrounded with 'X' cells if you can connect the region with 'X' cells and none of the region cells are on the edge of the `board`.

To capture a **surrounded region**, replace all 'O's with 'X's in-place within the original board. You do not need to return anything.

**Example 1:**

Input: `board = [ ["X","X","X","X"], ["X","O","O","X"], ["X","X","O","X"], ["X","O","X","X"] ]`

Output: `[ ["X","X","X","X"], ["X","X","X","X"], ["X","X","X","X"], ["X","O","X","X"] ]`

Explanation:

```
graph LR
    subgraph Input
        direction TB
        I1["X X X X"]
        I2["X O O X"]
        I3["X X O X"]
        I4["X O X X"]
    end
    subgraph Output
        direction TB
        O1["X X X X"]
        O2["X X X X"]
        O3["X X X X"]
        O4["X O X X"]
    end
```

136 Online

```
14 int r=board.size(),c=board[0].size();
15 for(int i=0;i<r;++i){
16     if(board[i][0]=='O') dfs(board, i, 0, r, c);
17     if(board[i][c-1]=='O') dfs(board, i, c-1, r, c);
18 }
19 for(int i=0;i<c;++i){
20     if(board[0][i]=='O') dfs(board, 0, i, r, c);
21     if(board[r-1][i]=='O') dfs(board, r-1, i, r, c);
22 }
23
24 for(int i=0;i<r;++i){
25     for(int j=0;j<c;++j){
26         if(board[i][j]=='O') board[i][j]='X';
27         else if(board[i][j]=='O') board[i][j]='X';
28     }
29 }
30
31 };
```

Accepted Runtime: 0 ms

Case 1 Case 2

Input

board = [ ["X","X","X","X"], ["X","O","O","X"], ["X","X","O","X"], ["X","O","X","X"] ]

### 4. Binary Tree Maximum Path Sum

124. Binary Tree Maximum Path Sum

Hard

A **path** in a binary tree is a sequence of nodes where each pair of adjacent nodes in the sequence has an edge connecting them. A node can only appear in the sequence **at most once**. Note that the path does not need to pass through the root.

The **path sum** of a path is the sum of the node's values in the path.

Given the `root` of a binary tree, return the **maximum path sum** of any **non-empty path**.

**Example 1:**

```
graph TD
    1((1)) --- 2((2))
    1 --- 3((3))
```

Input: `root = [1,2,3]`

Output: 6

Explanation: The optimal path is 2 -> 1 -> 3 with a path sum of 2 + 1 + 3 = 6.

259 Online

```
8
9 private:
10 int dfs(TreeNode* node, int& res) {
11     if (!node) {
12         return 0;
13     }
14
15     // Recursively compute the maximum sum of the left and right subtree paths.
16     int leftSum = max(0, dfs(node->left, res));
17     int rightSum = max(0, dfs(node->right, res));
18
19     // Update the maximum path sum encountered so far (with split).
20     res = max(res, leftSum + rightSum + node->val);
21
22     // Return the maximum sum of the path (without split).
23     return max(leftSum, rightSum) + node->val;
24 }
25 };
```

Accepted Runtime: 0 ms

Case 1 Case 2

Input

root = [1,2,3]