Documentation: Minesweeper (Leetcode 529)

1. Problem Statement

You are given an m x n matrix board representing a Minesweeper game board, where:

- 'M' represents an unrevealed mine.
- 'E' represents an unrevealed empty square.
- 'B' is a revealed blank square with no adjacent mines.
- '1'–'8' represents a revealed square with 1–8 adjacent mines.
- 'X' represents a revealed mine (game over).

You are also given a list click = [row, col] indicating a position to click. Your task is to update and return the board after revealing this cell, following these rules:

- If a mine ('M') is clicked, change it to 'X'.
- If an empty square ('E') with no adjacent mines is clicked, change it to 'B' and recursively reveal all adjacent unrevealed squares.
- If an empty square ('E') with at least one adjacent mine is clicked, change it to a digit ('1'–'8') showing the count of adjacent mines.
- No further action should be taken once revealing is complete.

2. Intuition

The game follows a recursive pattern where clicking on an empty cell may trigger a chain of reveals. If we reveal a blank square with no nearby mines, we must continue revealing its neighbors.

This makes a Depth-First Search (DFS) or Breadth-First Search (BFS) a natural fit for solving the problem.

3. Key Observations

- We only recurse if the square has no adjacent mines.
- The revealed number on a square equals the number of adjacent 'M'.

- There are 8 possible directions to check around a cell (up, down, left, right, and the four diagonals).
- No recursion is needed if a mine or a number is found.

4. Approach

Step-by-Step:

- Read the clicked cell coordinates.
- If it's a mine ('M'), convert it to 'X' and end the game.
- If it's an empty cell ('E'):
    1. Count adjacent mines.
    2. If no mines: set it to 'B' and DFS into all neighboring 'E' cells.
    3. If there are mines: set it to the corresponding number '1' to '8'.

DFS is used here for recursive exploration.

5. Edge Cases

- Click on a mine ('M') → return immediately with 'X'.
- The click is on an already revealed square → should not happen per constraints.
- All cells are already revealed → should not happen but handled gracefully.

6. Complexity Analysis

Time Complexity:

- Worst case: O(m × n) — if all cells are empty and connected (i.e., recursive reveals continue across the entire board).

Space Complexity:

- O(m × n) in the worst case due to recursion stack (DFS).

7. Alternative Approaches

- BFS (Queue-based) can be used instead of DFS to avoid recursion depth issues.
- BFS is also suitable for level-wise exploration and memory optimization.

8. Test Cases

✅ Test Case 1

```
board = [["E","E","E","E","E"],
         ["E","E","M","E","E"],
         ["E","E","E","E","E"],
         ["E","E","E","E","E"]]
click = [3, 0]
```

Output:

```
[["B","1","E","1","B"],
 ["B","1","M","1","B"],
 ["B","1","1","1","B"],
 ["B","B","B","B","B"]]
```

✅ Test Case 2

```
board = [["B","1","E","1","B"],
         ["B","1","M","1","B"],
         ["B","1","1","1","B"],
         ["B","B","B","B","B"]]
click = [1, 2]
```

Output:

```
[["B","1","E","1","B"],
 ["B","1","X","1","B"],
 ["B","1","1","1","B"],
 ["B","B","B","B","B"]]
```

9. Final Thoughts

- DFS is a clean and intuitive approach to this problem, but in large boards, switching to BFS may prevent stack overflows.
- The problem demonstrates the importance of understanding multi-directional traversal in matrix-based problems.
- Remember to handle base cases like clicking on a mine and preventing repeated visits to already revealed cells.