LeetCode 529

https://leetcode.com/problems/minesweeper/description/

Yifeng Zeng

Description

529. Minesweeper

Idea Report

Given a board matrix and a clicking point we need to update this board by some rules:

- 1. If the click point is an unrevealed mine 'M', we change it to revealed mine 'X' and no further step will be needed.
- 2. If the click point is an already revealed square no matter it's 'M'/'B'/'X'/'Digit', we just return the same board because it is a click that doesn't do anything.
- 3. If the click point is an unrevealed empty square 'E', we need to do two different things depends on what that square should be.
  - 3.1. If it should be a digit, we change it form 'E' to the 'Digit'
  - 3.2. If it should be a revealed blank square 'B', we change it to 'B' and do a search to find all the neighboring 'B'. We stop the searching if we find an already revealed square. Or we stop if we find a square that should be a 'Digit', and also we need to change that sqaure to the 'Digit'.

We can use both BFS or DFS to do the search.

Code

```
class Solution {
    //AC BFS
    final int[] dx = {0,  0, 1, -1, 1,  1, -1, -1};
    final int[] dy = {1, -1, 0,  0, 1, -1,  1, -1};

    public char[][] updateBoard(char[][] board, int[] click) {
        final int rows = board.length;
        final int cols = board[0].length;
```

```
        int x = click[0];
        int y = click[1];
        if (board[x][y] == 'M') {
            board[x][y] = 'X';
            return board;
        }
        if (board[x][y] != 'E') {
            return board;
        }

        int count = check(board, click[0], click[1]);
        if (count != 0) {
            board[x][y] = (char) (count + '0');
            return board;
        }

        // BFS
        Deque<int[]> q = new LinkedList<>();
        q.offer(click);
        board[x][y] = 'B';

        while (!q.isEmpty()) {
            int[] cur = q.poll();
            for (int i = 0; i < dx.length; i++) {
                int r = cur[0] + dx[i];
                int c = cur[1] + dy[i];
                if (!isValid(board, r, c) || board[r][c] != 'E') {
                    continue;
                }
                count = check(board, r, c);
                if (count == 0) {
                    q.offer(new int[]{r, c});
                    board[r][c] = 'B';
                } else {
                    board[r][c] = (char) (count + '0');
                }
            }
        }

        return board;
    }


    private int check(char[][] board, int row, int col) {
        int count = 0;
        for (int i = 0; i < dx.length; i++) {
            int r = row + dx[i];
            int c = col + dy[i];
            if (isValid(board, r, c) && board[r][c] == 'M') {
                count++;
            }
```

```
        }
        return count;
    }


    private boolean isValid(char[][] board, int r, int c) {
        if (0 <= r && r < board.length && 0 <= c && c < board[0].length) {
            return true;
        }
        return false;
    }
}
```

Code

```
class Solution {

    // AC DFS
    final int[] dx = {0,  0, 1, -1, 1,  1, -1, -1};
    final int[] dy = {1, -1, 0,  0, 1, -1,  1, -1};

    public char[][] updateBoard(char[][] board, int[] click) {
        dfsHelper(board, click);
        return board;
    }

    private void dfsHelper(char[][] board, int[] click) {
        int x = click[0];
        int y = click[1];
        if (!isValid(board, x, y)) {
            return;
        }

        if (board[x][y] == 'M') {
            board[x][y] = 'X';
            return;
        }
        if (board[x][y] != 'E') {
            return;
        }
        int count = check(board, click[0], click[1]);
        if (count != 0) {
            board[x][y] = (char) (count + '0');
            return;
        }

        board[x][y] = 'B';
        for (int i = 0; i < dx.length; i++) {
```

```
                click[0] = x + dx[i];
                click[1] = y + dy[i];
                dfsHelper(board, click);
            }
        }


        private int check(char[][] board, int row, int col) {
            int count = 0;
            for (int i = 0; i < dx.length; i++) {
                int r = row + dx[i];
                int c = col + dy[i];
                if (isValid(board, r, c) && board[r][c] == 'M') {
                    count++;
                }
            }
            return count;
        }


        private boolean isValid(char[][] board, int r, int c) {
            if (0 <= r && r < board.length && 0 <= c && c < board[0].length) {
                return true;
            }
            return false;
        }
    }
```

Summary

- Use int[] dx, int[] dy to simplify the search direction.
- Analysing the steps first then the coding part is just following the steps.
- Try to modulize the code, like isValid() etc.