题目：

给你一个由 `'1'`（陆地）和 `'0'`（水）组成的的二维网格，请你计算网格中岛屿的数量。

岛屿总是被水包围，并且每座岛屿只能由水平方向或竖直方向上相邻的陆地连接形成。

此外，你可以假设该网格的四条边均被水包围。

示例 1：

输入：
[
['1','1','1','1','0'],
['1','1','0','1','0'],
['1','1','0','0','0'],
['0','0','0','0','0']
]
输出： 1

示例 2：

输入：
[
['1','1','0','0','0'],
['1','1','0','0','0'],
['0','0','1','0','0'],
['0','0','0','1','1']
]
输出： 3
解释： 每座岛屿只能由水平和/或竖直方向上相邻的陆地连接而成。

思路：

1 递归：

遍历数组，如果发现1，就把岛屿数量 + 1，把其周围（左右，上下）的都1都变为0

```
def numIslands(self, grid):
        """
        :type grid: List[List[str]]
        :rtype: int
        """

        def dfs( grid, r, c):
            grid[r][c] = 0
            nr, nc = len(grid), len(grid[0])
             for x, y in [(r - 1, c), (r + 1, c), (r, c - 1), (r, c + 1)]
                 if 0 <= x < nr and 0 <= y < nc and grid[x][y] == "1":
                     dfs(grid, x, y)

         # 长
         nw = len(grid)
```

```
        if nw == 0:
            return 0

        nh = len(grid[0])

        # 设置岛屿数量:
        num_island = 0
        for w in range(nw):
            for h in range(nh):
                if grid[w][h] == "1":
                    num_island += 1
                    dfs(grid,w, h)

        return num_island
```

2 迭代:

```
def numIslands(self, grid):
        """
        :type grid: List[List[str]]
        :rtype: int
        """
        # 行
        nh = len(grid)
        if nh == 0:
            return 0
        # 列
        nw = len(grid[0])

        num_island = 0
        for i in range(nh):
            for j in range(nw):
                if grid[i][j] == "1":
                    num_island += 1
                    # 把岛屿周围1都清0
                    grid[i][j] = "0"
                    neighbors = collections.deque([(i, j)])
                    while neighbors:
                        row, col = neighbors.popleft()
                        for x, y in [(row - 1, col), (row + 1, col), (ro
                            if 0 <= x < nh and 0 <= y < nw and grid[x][y
                                neighbors.append((x, y))
                                grid[x][y] = "0"
```

```
27          return num_island
28
```

扫雷游戏：

让我们一起来玩扫雷游戏！

给定一个代表游戏板的二维字符矩阵。 **'M'** 代表一个**未挖出的**地雷，**'E'** 代表一个**未挖出的**空方块，**'B'** 代表没有相邻（上，下，左，右，和所有4个对角线）地雷的**已挖出的**空白方块，**数字**（'1' 到 '8'）表示有多少地雷与这块**已挖出的**方块相邻，**'X'** 则表示一个**已挖出的**地雷。

现在给出在所有**未挖出的**方块中（'M'或者'E'）的下一个点击位置（行和列索引），根据以下规则，返回相应位置被点击后对应的面板：

1. 如果一个地雷（'M'）被挖出，游戏就结束了- 把它改为 **'X'**。
2. 如果一个**没有相邻地雷**的空方块（'E'）被挖出，修改它为（'B'），并且所有和其相邻的**未挖出**方块都应该被递归地揭露。
3. 如果一个**至少与一个地雷相邻**的空方块（'E'）被挖出，修改它为数字（'1'到'8'），表示相邻地雷的数量。
4. 如果在此次点击中，若无更多方块可被揭露，则返回面板。

示例 1：

输入：

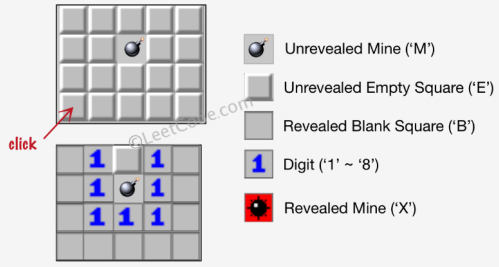
```
[['E', 'E', 'E', 'E', 'E'],
 ['E', 'E', 'M', 'E', 'E'],
 ['E', 'E', 'E', 'E', 'E'],
 ['E', 'E', 'E', 'E', 'E']]

Click : [3,0]
```

输出：

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

解释：



示例 2：

输入：

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

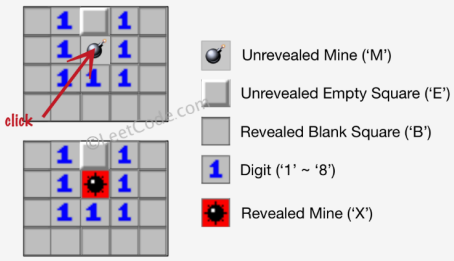
```
 ['B', '1', '1', '1', 'B'],
 ['B', 'B', 'B', 'B', 'B']]

Click : [1,2]
```

输出：

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

解释：



注意：

1. 输入矩阵的宽和高的范围为 [1,50]。
2. 点击的位置只能是未被挖出的方块 ('M' 或者 'E')，这也意味着面板至少包含一个可点击的方块。
3. 输入面板不会是游戏结束的状态（即有地雷已被挖出）。
4. 简单起见，未提及的规则在这个问题中可被忽略。例如，当游戏结束时你不需要挖出所有地雷，考虑所有你可能赢得游戏或标记方块的情况。

DFS：注意，这里需要考虑8个方向

```python
def updateBoard(self, board, click):
        """
        :type board: List[List[str]]
        :type click: List[int]
        :rtype: List[List[str]]
        """
        i, j = click
        row, col = len(board),len(board[0])
        if board[i][j] == 'M':
            board[i][j] = 'X'
            return board
         res = 0
         # 计算地雷的数量
         def cal(i, j):
            res = 0
            # 8个方向
            for x in [1, -1, 0]:
                for y in [1, -1, 0]:
```

```
                    if x == 0 and y == 0:
                        continue
                    if 0 <= i + x < row and 0 <= j + y < col and board[i
                        res += 1

            return res

        # 将E换成 B:
        def dfs(i, j):
            num = cal(i, j)
            if num > 0:
                board[i][j] = str(num)
                return
            board[i][j] = "B"
            for x in [1, -1, 0]:
                for y in [1, -1, 0]:
                    if x == 0 and y == 0:
                        continue
                    if 0 <= i + x < row and 0 <= j + y < col and board[i
                        dfs(i+x, j+y)
        dfs(i,j)
        return board
```

```
def updateBoard(self, board, click):
        """
        :type board: List[List[str]]
        :type click: List[int]
        :rtype: List[List[str]]
        """
        dx = [-1, 1, 0, 0, -1, 1, -1, 1]
        dy = [0, 0, -1, 1, -1, 1, 1, -1]

        x, y = click
        if board[x][y] == 'M':
            board[x][y] = 'X'
            return board

        def dfs(board, x, y):
            row, col = len(board), len(board[0])
             # 递归终止条件，空地(i,j)是否有雷，有，则吧周边位置修改为雷数，终止该
            res = 0
```

```
            # 计算雷数;
            for k in range(8):
                i, j = x + dx[k], y + dy[k]
                if 0 <= i < row and 0 <= j < col and board[i][j] == "M":
                    res += 1

            if res > 0:
                board[x][y] = str(res)
                return

            # 若没有雷，则将该位置置为“B”，向 8 邻域的空地继续搜索。
            board[x][y] = "B"
            for k in range(8):
                i, j = x + dx[k], y + dy[k]
                if 0 <= i < row and 0 <= j < col and board[i][j] == "E":
                    dfs(board, i, j )


        dfs(board,x,y)
        return board
```

BFS :