130. Surrounded Regions

题目描述：https://leetcode.com/problems/surrounded-regions/

> 给定一个棋盘，判断X包含的O是否需要被吃掉。返回把所有能吃到的O吃掉的结果。
> 例如：
>
> ```
> For example,
> X X X X
> X O O X
> X X O X
> X O X X
> After running your function, the board should be:
> X X X X
> X X X X
> X X X X
> X O X X
> ```

解题思路：

1. BFS 挨着边边的O可以被留下，被该O可达的O可以被留下，其他被吃掉！
   因此从边边上的O开始BFS就可以了
2. UNION FIND

代码 **BFS**：

```
class Solution {
public:
    void solve(vector<vector<char>>& board) {
        int m = board.size();
        if(m == 0) return ;
        int n = board[0].size();
        if(n == 0) return ;
        for(int i = 0; i < m; i++) {
            BFS(board, i, 0, m, n);
            BFS(board, i, n-1, m, n);
        }
        for(int j = 0; j < n; j++) {
            BFS(board, 0, j, m, n);
            BFS(board, m-1, j, m, n);
        }
        for(int i = 0; i < m; i++) {
            for(int j = 0; j < n; j++) {
                if(board[i][j] == 'O') board[i][j] = 'X';
                if(board[i][j] == '*') board[i][j] = 'O';
            }
        }
        return;
    }
    void BFS(vector<vector<char> >& board, int i, int j, int m, int n) {
        if(board[i][j] == 'X') {
            return ;
        }
        if(board[i][j] == 'O') {
            board[i][j] = '*';
            if(i > 1) BFS(board, i-1, j, m, n);
            if(i < m-2) BFS(board, i+1, j, m, n);
            if(j > 1) BFS(board, i, j-1, m, n);
            if(j < n-2) BFS(board, i, j+1, m, n);
        }
    }
};
```

代码UnionFind：

```
class Solution {
public:
    vector<int> p;
    void makeset(vector<vector<char>>& board) {
        int m = board.size();
        int n = board[0].size();
        p.resize(m*n+1);
        for(int i = 0; i < m; i++) {
```

```cpp
            for(int j = 0; j < n; j++) {
                p[i*n+j] = i*n+j;
            }
        }
        p[m*n] = m*n;
        return;
    }
    int findFather(int i) {
        if(i == p[i]) return i;
        int t = findFather(p[i]);
        p[i] = t;
        return t;
    }
    void myUnion(int i, int j) {
        int pi = findFather(i);
        int pj = findFather(j);
        if(pi == pj) return;
        p[pi] = pj;
        return;
    }
    void solve(vector<vector<char>>& board) {
        int m = board.size();
        if(m == 0) return;
        int n = board[0].size();
        makeset(board);
        for(int i = 0; i < m; i++) {
            for(int j = 0; j < n; j++) {
                if(board[i][j] == 'X') continue;
                int a = i*n+j;
                if(i == 0|| i == m-1 ||j == 0||j == n-1) {
                    myUnion(a, m*n);
                }
                if(i+1 < m && board[i+1][j] == 'O') myUnion(a, a+n);
                if(i-1 >= 0 && board[i-1][j] == 'O') myUnion(a, a-n);
                if(j+1 < n && board[i][j+1] == 'O') myUnion(a, a+1);
                if(j-1 >= 0 && board[i][j-1] == 'O') myUnion(a, a-1);
            }
        }
        for(int i = 1; i < m-1; i++) {
            for(int j = 1; j < n-1; j++) {
                if(board[i][j] == 'O' && findFather(i*n+j) != findFather(m*n)) {
                    board[i][j] = 'X';
                }
            }
        }
        return;
    }
};
```