Surrounded Regions

Given a 2D board containing `'X'` and `'O'`, capture all regions surrounded by `'X'`.

A region is captured by flipping all `'O'`s into `'X'`s in that surrounded region.

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
```

Solution 1

The algorithm is quite simple: Use BFS starting from 'O's on the boundary and mark them as 'B', then iterate over the whole board and mark 'O' as 'X' and 'B' as 'O'.

```
void bfsBoundary(vector<vector<char> >& board, int w, int l)
{
    int width = board.size();
    int length = board[0].size();
    deque<pair<int, int> > q;
    q.push_back(make_pair(w, l));
    board[w][l] = 'B';
    while (!q.empty()) {
        pair<int, int> cur = q.front();
        q.pop_front();
        pair<int, int> adjs[4] = {{cur.first-1, cur.second},
            {cur.first+1, cur.second},
            {cur.first, cur.second-1},
            {cur.first, cur.second+1}};
        for (int i = 0; i < 4; ++i)
        {
            int adjW = adjs[i].first;
            int adjL = adjs[i].second;
            if ((adjW >= 0) && (adjW < width) && (adjL >= 0)
                    && (adjL < length)
                    && (board[adjW][adjL] == 'O')) {
                q.push_back(make_pair(adjW, adjL));
                board[adjW][adjL] = 'B';
            }
        }
    }
}

void solve(vector<vector<char> > &board) {
    int width = board.size();
    if (width == 0) //Add this to prevent run-time error!
        return;
    int length = board[0].size();
    if  (length == 0) // Add this to prevent run-time error!
        return;

    for (int i = 0; i < length; ++i)
    {
        if (board[0][i] == 'O')
            bfsBoundary(board, 0, i);

        if (board[width-1][i] == 'O')
            bfsBoundary(board, width-1, i);
    }

    for (int i = 0; i < width; ++i)
    {
        if (board[i][0] == 'O')
            bfsBoundary(board, i, 0);
        if (board[i][length-1] == 'O')
            bfsBoundary(board, i, length-1);
```

```
    }

    for (int i = 0; i < width; ++i)
    {
        for (int j = 0; j < length; ++j)
        {
            if (board[i][j] == 'O')
                board[i][j] = 'X';
            else if (board[i][j] == 'B')
                board[i][j] = 'O';
        }
    }
}
```

Note that one of the test cases is when the board is empty. So if you don't check it in your code, you will encounter an run-time error.

written by eaglesky1990 original link here

Solution 2

```
class UF
{
private:
    int* id;     // id[i] = parent of i
    int* rank;  // rank[i] = rank of subtree rooted at i (cannot be more than 31)
    int count;    // number of components
public:
    UF(int N)
    {
        count = N;
        id = new int[N];
        rank = new int[N];
        for (int i = 0; i < N; i++) {
            id[i] = i;
            rank[i] = 0;
        }
    }
    ~UF()
    {
        delete [] id;
        delete [] rank;
    }
    int find(int p) {
        while (p != id[p]) {
            id[p] = id[id[p]];    // path compression by halving
            p = id[p];
        }
        return p;
    }
    int getCount() {
        return count;
    }
    bool connected(int p, int q) {
        return find(p) == find(q);
    }
    void connect(int p, int q) {
        int i = find(p);
        int j = find(q);
        if (i == j) return;
        if (rank[i] < rank[j]) id[i] = j;
        else if (rank[i] > rank[j]) id[j] = i;
        else {
            id[j] = i;
            rank[i]++;
        }
        count--;
    }
};

class Solution {
public:
    void solve(vector<vector<char>> &board) {
        int n = board.size();
        if(n==0)    return;
```

```
        int m = board[0].size();
        UF uf = UF(n*m+1);

        for(int i=0;i<n;i++){
            for(int j=0;j<m;j++){
                if((i==0||i==n-1||j==0||j==m-1)&&board[i][j]=='O') // if a 'O' no
de is on the boundry, connect it to the dummy node
                    uf.connect(i*m+j,n*m);
                else if(board[i][j]=='O') // connect a 'O' node to its neighbour
'O' nodes
                {
                    if(board[i-1][j]=='O')
                        uf.connect(i*m+j,(i-1)*m+j);
                    if(board[i+1][j]=='O')
                        uf.connect(i*m+j,(i+1)*m+j);
                    if(board[i][j-1]=='O')
                        uf.connect(i*m+j,i*m+j-1);
                    if(board[i][j+1]=='O')
                        uf.connect(i*m+j,i*m+j+1);
                }
            }
        }

        for(int i=0;i<n;i++){
            for(int j=0;j<m;j++){
                if(!uf.connected(i*m+j,n*m)){ // if a 'O' node is not connected t
o the dummy node, it is captured
                    board[i][j]='X';
                }
            }
        }
    }
};
```

Hi. So here is my accepted code using **Union Find** data structure. The idea comes from the observation that if a region is NOT captured, it is connected to the boundry. So if we connect all the 'O' nodes on the boundry to a dummy node, and then connect each 'O' node to its neighbour 'O' nodes, then we can tell directly whether a 'O' node is captured by checking whether it is connected to the dummy node. For more about Union Find, the first assignment in the algo1 may help: https://www.coursera.org/course/algs4partI

written by jinming.he.5 original link here

Solution 3

- First, check the four border of the matrix. If there is a element is 'O', alter it and all its neighbor 'O' elements to '1'.
- Then ,alter all the 'O' to 'X'
- At last,alter all the '1' to 'O'

For example:

```
X X X X           X X X X             X X X X
X X O X  ->       X X O X    ->       X X X X
X O X X           X 1 X X             X O X X
X O X X           X 1 X X             X O X X
```

```
class Solution {
public:
    void solve(vector<vector<char>>& board) {
        int i,j;
        int row=board.size();
        if(!row)
            return;
        int col=board[0].size();

        for(i=0;i<row;i++){
            check(board,i,0,row,col);
            if(col>1)
                check(board,i,col-1,row,col);
        }
        for(j=1;j+1<col;j++){
            check(board,0,j,row,col);
            if(row>1)
                check(board,row-1,j,row,col);
        }
        for(i=0;i<row;i++)
            for(j=0;j<col;j++)
                if(board[i][j]=='O')
                    board[i][j]='X';
        for(i=0;i<row;i++)
            for(j=0;j<col;j++)
                if(board[i][j]=='1')
                    board[i][j]='O';
    }
    void check(vector<vector<char> >&vec,int i,int j,int row,int col){
        if(vec[i][j]=='O'){
            vec[i][j]='1';
            if(i>1)
                check(vec,i-1,j,row,col);
            if(j>1)
                check(vec,i,j-1,row,col);
            if(i+1<row)
                check(vec,i+1,j,row,col);
            if(j+1<col)
                check(vec,i,j+1,row,col);
        }
    }
};
```

written by sugeladi original link here