Contain Virus

A virus is spreading rapidly, and your task is to quarantine the infected area by installing walls.

The world is modeled as a 2-D array of cells, where `0` represents uninfected cells, and `1` represents cells contaminated with the virus. A wall (and only one wall) can be installed **between any two 4-directionally adjacent cells**, on the shared boundary.

Every night, the virus spreads to all neighboring cells in all four directions unless blocked by a wall. Resources are limited. Each day, you can install walls around only one region -- the affected area (continuous block of infected cells) that threatens the most uninfected cells the following night. There will never be a tie.

Can you save the day? If so, what is the number of walls required? If not, and the world becomes fully infected, return the number of walls used.

Example 1:

```
Input: grid =
[[0,1,0,0,0,0,0,1],
 [0,1,0,0,0,0,0,1],
 [0,0,0,0,0,0,0,1],
 [0,0,0,0,0,0,0,0]]
Output: 10
Explanation:
There are 2 contaminated regions.
On the first day, add 5 walls to quarantine the viral region on the left. The board a
fter the virus spreads is:

[[0,1,0,0,0,0,1,1],
 [0,1,0,0,0,0,1,1],
 [0,0,0,0,0,0,1,1],
 [0,0,0,0,0,0,0,1]]

On the second day, add 5 walls to quarantine the viral region on the right. The virus
 is fully contained.
```

Example 2:

```
Input: grid =
[[1,1,1],
 [1,0,1],
 [1,1,1]]
Output: 4
Explanation: Even though there is only one cell saved, there are 4 walls built.
Notice that walls are only built on the shared boundary of two different cells.
```

Example 3:

```
Input: grid =
[[1,1,1,0,0,0,0,0,0],
 [1,0,1,0,1,1,1,1,1],
 [1,1,1,0,0,0,0,0,0]]
Output: 13
Explanation: The region on the left only builds two new walls.
```

Note:

1. The number of rows and columns of `grid` will each be in the range `[1, 50]`.
2. Each `grid[i][j]` will be either `0` or `1`.
3. Throughout the described process, there is always a contiguous viral region that will infect **strictly more** uncontaminated squares in the next round.

Solution 1

Not sure do I need to match this in my code.

written by cshshshzx original link here

Solution 2

The solution simply models the process.

1. Build walls = set those connected virus inactive, i.e. set as -1;
2. Affected area != walls; For example, one 0 surrounded by all 1s have area = 1, but walls = 4.

DFS

```
class Solution {
public:
    int containVirus(vector<vector<int>>& grid) {
        int ans = 0;
        while (true) {
            int walls = process(grid);
            if (walls == 0) break; // No more walls to build
            ans += walls;
        }
        return ans;
    }
private:
    int process(vector<vector<int>>& grid) {
        int m = grid.size(), n = grid[0].size();
        // cnt is max area to be affected by a single virus region; ans is correspon
ding walls
        int cnt = 0, ans = 0, color = -1, row = -1, col = -1;
        // visited virus as 1, visited 0 using different color to indicate being aff
ected by different virus
        vector<vector<int>> visited(m, vector<int>(n, 0));
        for (int i = 0; i < m; i++) {
            for (int j = 0; j < n; j++) {
                if (grid[i][j] == 1 && visited[i][j] == 0) {
                    int walls = 0, area = dfs(grid, visited, i, j, color, walls);
                    if (area > cnt) {
                        ans = walls;
                        cnt = area;
                        row = i;
                        col = j;
                    }
                    color--;
                }
            }
        }
        // set this virus region inactive
        buildWall(grid, row, col);
        // propagate other virus by 1 step
        visited = vector<vector<int>>(m, vector<int>(n, 0));
        for (int i = 0; i < m; i++) {
            for (int j = 0; j < n; j++) {
                if (grid[i][j] == 1 && visited[i][j] == 0)
                    spread(grid, visited, i, j);
            }
        }
        return ans;
    }
```

```
    int dfs(vector<vector<int>>& grid, vector<vector<int>>& visited, int row, int co
l, int color, int& walls) {
        int m = grid.size(), n = grid[0].size(), ans = 0;
        if (row < 0 || row >= m || col < 0 || col >= n) return 0;
        if (grid[row][col] == 0) {
            walls++;
            if (visited[row][col] == color) return 0;
            visited[row][col] = color;
            return 1;
        }
        // grid[row][col] could be -1, inactive virus
        if (visited[row][col] == 1 || grid[row][col] != 1) return 0;
        visited[row][col] = 1;
        vector<int> dir = {-1, 0, 1, 0, -1};
        for (int i = 0; i < 4; i++)
            ans += dfs(grid, visited, row+dir[i], col+dir[i+1], color, walls);
        return ans;
    }
    void buildWall(vector<vector<int>>& grid, int row, int col) {
        int m = grid.size(), n = grid[0].size();
        if (row < 0 || row >= m || col < 0 || col >= n || grid[row][col] != 1) retur
n;
        grid[row][col] = -1; //set inactive
        vector<int> dir = {-1, 0, 1, 0, -1};
        for (int i = 0; i < 4; i++)
            buildWall(grid, row+dir[i], col+dir[i+1]);
    }
    void spread(vector<vector<int>>& grid, vector<vector<int>>& visited, int row, in
t col) {
        int m = grid.size(), n = grid[0].size();
        if (row < 0 || row >= m || col < 0 || col >= n || visited[row][col] == 1) re
turn;
        if (grid[row][col] == 0) {
            grid[row][col] = 1;
            visited[row][col] = 1;
        }
        else if (grid[row][col] == 1) {
           visited[row][col] = 1;
           vector<int> dir = {-1, 0, 1, 0, -1};
           for (int i = 0; i < 4; i++)
               spread(grid, visited, row+dir[i], col+dir[i+1]);
        }
    }
};
```

DFS, single pass with intermediate results saved, 19 ms

```
class Solution {
public:
    int containVirus(vector<vector<int>>& grid) {
        int ans = 0;
        while (true) {
            int walls = model(grid);
            if (walls == 0) break;
            ans += walls;
        }
```

```
        return ans;
    }
private:
    int model(vector<vector<int>>& grid) {
        int m = grid.size(), n = grid[0].size(), N = 100;
        vector<unordered_set<int>> virus, toInfect;
        vector<vector<int>> visited(m, vector<int>(n, 0));
        vector<int> walls;
        for (int i = 0; i < m; i++) {
            for (int j = 0; j < n; j++) {
                if (grid[i][j] == 1 && visited[i][j] == 0) {
                    virus.push_back(unordered_set<int>());
                    toInfect.push_back(unordered_set<int>());
                    walls.push_back(0);
                    dfs(grid, visited, virus.back(), toInfect.back(), walls.back(),
i, j);
                }
            }
        }
        int maxArea = 0, idx = -1;
        for (int i = 0; i < toInfect.size(); i++) {
            if (toInfect[i].size() > maxArea) {
                maxArea = toInfect[i].size();
                idx = i;
            }
        }
        if (idx == -1) return 0;
        for (int i = 0; i < toInfect.size(); i++) {
            if (i != idx) {
                for (int key : toInfect[i])
                    grid[key/N][key%N] = 1;
            }
            else {
                for (int key: virus[i])
                    grid[key/N][key%N] = -1;
            }
        }
        return walls[idx];
    }
private:
    void dfs(vector<vector<int>>& grid, vector<vector<int>>& visited, unordered_set<
int>& virus, unordered_set<int>& toInfect, int& wall, int row, int col) {
        int m = grid.size(), n = grid[0].size(), N = 100;
        if (row < 0 || row >= m || col < 0 || col >= n || visited[row][col] == 1) re
turn;
        if (grid[row][col] == 1) {
            visited[row][col] = 1;
            virus.insert(row*N + col);
            vector<int> dir = {0, -1, 0, 1, 0};
            for (int i = 0; i < 4; i++)
                dfs(grid, visited, virus, toInfect, wall, row+dir[i], col+dir[i+1])
;
        }
        else if (grid[row][col] == 0) {
            wall++;
            toInfect.insert(row*N + col);
        }
```

```
    }
};
```

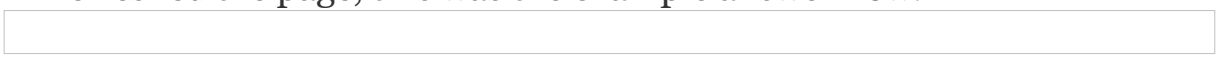
written by zestypanda original link here

Solution 3

This is what I saw during the contest, spent lots of time figuring out the logic (guess it's 11 because we can't overlap the walls)

But by the time I submitted my answer, it was wrong!

When I refreshed the page, this was the example answer now.



written by cshshshzx original link here