749. Contain Virus

DescriptionHintsSubmissionsDiscussSolution

- Difficulty:Hard
- Total Accepted:359
- Total Submissions:1K
- Contributor:yujiehuang

Related Topic :Depth First Search

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 after 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.

Seen this question in a real interview before?

```
#include<stdio.h>
#include<iostream>
#include<sstream>
#include<vector>
#include<algorithm>
#include<unordered_map>
#include<limits.h>
#include<queue>
#include<unordered_map>
#include<unordered_set>
#include<stack>
#include<string.h>
using namespace std;

int dfs(vector<vector<int>> &grid, vector<vector<int>> &visited, int row, int 
col, int color,
int &walls)
{
      int m = grid.size();
      int n = grid[0].size();
      int ans = 0;
      if(row>=m || col>=n || row<0 || col<0) return 0;
      if(grid[row][col]==0)
      {
            walls++;
            if(visited[row][col]==color) return 0;
            visited[row][col]=color;
            return 1;
      }
      // we have visited an inactive point, skip it
      if(visited[row][col]==1 || grid[row][col]!=1) return 0;
      visited[row][col]=1;
      vector<vector<int>> directions = {{1,0},{-1,0},{0,1},{0,-1}};
      for(auto elem:directions)
      {
            ans+=dfs(grid,visited,row+elem[0],col+elem[1],color,walls);
      }
      return ans;
}

void buildWall(vector<vector<int>> &grid, int row, int col)
{
      int m = grid.size();
      int n = grid[0].size();
      if(row>=m || col>=n || row<0 || col<0 || grid[row][col]!=1 ) return;
      //set as inactive
      grid[row][col]=-1;
      vector<vector<int>> directions = {{1,0},{-1,0},{0,1},{0,-1}};
      for(auto elem:directions)
      {
            buildWall(grid,row+elem[0],col+elem[1]);
      }
}

void spread(vector<vector<int>> &grid, vector<vector<int>> &visited, int row,
int col)
{
      int m = grid.size();
      int n = grid[0].size();
      if(row>=m || col>=n || row<0 || col<0 || visited[row][col]==1) return;
      if(grid[row][col]==0)
      {
            grid[row][col]=1;
            visited[row][col]=1;
      }else if (grid[row][col]==1){
            visited[row][col]=1;
            vector<vector<int>> directions = {{1,0},{-1,0},{0,1},{0,-1}};
            for(auto elem:directions)
            {
                  spread(grid,visited,row+elem[0],col+elem[1]);
            }
      }
}

int process(vector<vector<int>> &grid)
{
      int m = grid.size();
      int n = grid[0].size();
      int color = -1;
      vector<vector<int>> visited(m,vector<int>(n,0));
      int row,col; int max_area = INT_MIN;
      int area = 0; int ans = 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>max_area)
                        {
                              max_area = area;
                              col = j;
                              row = i;
```

```
                    ans = walls;
                }
            }
            color--;
        }
    }

    // build the wall near the max area connected region
    buildWall(grid,row,col);
    // spread the virus of others
    // clear the visited
    for(int i=0;i<m;++i)
    {
        fill_n(visited[i].begin(),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 containVirus(vector<vector<int>>& grid) {
    int ans = 0;
    while(true)
    {
        int wall = process(grid);
        cout << wall << endl;
        if(wall==0) break;
        ans+=wall;
    }
    return ans;
}

int main(int argc, char *argv[])
{
    vector<vector<int>> 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}};
    int ans = containVirus(grid);
    cout << ans << endl;
    return 0;
}
```