Горски Пожари

locked

Горските пожари са сериозен проблем. Помните Австралия.

Разполагате с квадратна карта на гора. С 0 са обозначени дървета, които са добре а с 1 - горящи такива. За гасенето на пожарите са необходими много литри вода. Правителството обаче има нужда от помощта ви. Тъй като дърветата не могат да бъдат гасени едно по едно, от вас се иска да кажете какво е лицето на обграждащата площ (bounding box) на всеки горящ регион от гората.

Input Format

На първият ред прочитате цяло число N

Следва гората, представена като квадратна матрица със страна N.

Гарантирано е, че в матрицата има поне 1 клетка с `1`.

Constraints

$1 \leq N \leq 4000$

Output Format

На един ред на стандартния изход, разделени с интервал, изведете в намаляващ ред площите на всеки от споменатите региони. Виж примера за по добра визуализация.

Note: Приемете, че покриващите площи не се припокриват.

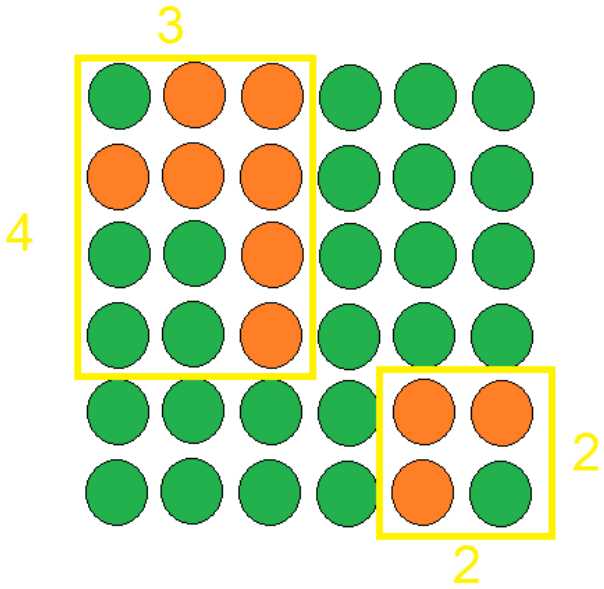
Sample Input 0

```
6
0 1 1 0 0 0
1 1 1 0 0 0
0 0 1 0 0 0
0 0 1 0 0 0
0 0 0 0 1 1
0 0 0 0 1 0
```

Sample Output 0

```
12 4
```

Explanation 0



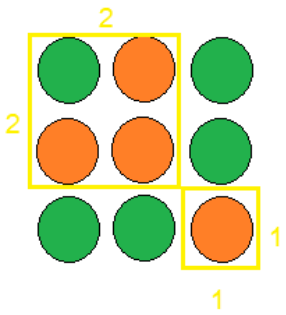
Sample Input 1

```
3
0 1 0
1 1 0
0 0 1
```

Sample Output 1

```
4 1
```

Explanation 1



Обърнете внимание, че дърветата не са съседни по диагонал.

Submissions: 69
Max Score: 100
Difficulty: Medium

Rate This Challenge:

More

Current Buffer (saved locally, editable) C++14

```
#include <cmath>
#include <cstdio>
#include <vector>
#include <iostream>
#include <algorithm>
#include <queue>
#include <climits>
```

```cpp
#include <set>
using namespace std;

int** arr;
bool** visited;
priority_queue<int> ans;
deque<pair<int, int>> q; //(x, y)
vector<int> areas;
set<pair<int, int>> qSet;

int main() {
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);

    int n;
    cin >> n;

    //initialize and fill
    arr = new int* [n];
    visited = new bool* [n];
    for (int i = 0; i < n; i++) {
        arr[i] = new int[n];
        visited[i] = new bool[n];
        for (int j = 0; j < n; j++) {
            cin >> arr[i][j];
            visited[i][j] = 0;
        }
    }

    //print
    /*cout << endl;
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            cout<<arr[i][j]<<" ";
        }
        cout << endl;
    }*/

    int minX = INT_MAX, minY = INT_MAX, maxX = -1, maxY = -1;
    bool changed = false;

    //traverse matrix
    q.push_back({ 0,0 });
    qSet.insert({ 0,0 });

    while (!q.empty()) {
        int currX = q.front().first;
        int currY = q.front().second;

        if (currX > n || currY > n)
            break;
        visited[currX][currY] = 1;

        //updating minX,minY,maxX,maxY if neccesary
        if (currX < minX)
            minX = currX;
        if (currY < minY)
            minY = currY;
        if (currX > maxX)
            maxX = currX;
        if (currY > maxY)
            maxY = currY;

        //checking all 4 neighboors and adding those who are not in the queue

        //above
        if (currX >= 0 && currX < n && currY - 1 >= 0 && currY - 1 < n) {//valid coords check
            auto it = qSet.find({ currX,currY - 1 });
            if (arr[currX][currY - 1] && (it == qSet.end())) { //check if el is 1 and is not in
queue
                q.push_back({ currX,currY - 1 });
                qSet.insert({ currX,currY - 1 });
            }
        }

```

```
        //bottom
        if (currX >= 0 && currX < n && currY + 1 >= 0 && currY + 1 < n) {//valid coords check
            auto it = qSet.find({ currX,currY + 1 });
            if (arr[currX][currY + 1] && (it == qSet.end())) { //check if el is 1 and is not in queue
                q.push_back({ currX,currY + 1 });
                qSet.insert({ currX,currY + 1 });
            }
        }

        //left
        if (currX - 1 >= 0 && currX - 1 < n && currY >= 0 && currY < n) {//valid coords check
            auto it = qSet.find({ currX - 1,currY });
            if (arr[currX - 1][currY] && (it == qSet.end())) { //check if el is 1 and is not in queue
                q.push_back({ currX - 1,currY });
                qSet.insert({ currX - 1,currY });
            }
        }

        //right
        if (currX + 1 >= 0 && currX + 1 < n && currY >= 0 && currY < n) {//valid coords check
            auto it = qSet.find({ currX + 1,currY });
            if (arr[currX + 1][currY] && (it == qSet.end())) { //check if el is 1 and is not in queue
                q.push_back({ currX + 1,currY });
                qSet.insert({ currX + 1,currY });
            }
        }

        q.pop_front();

        //check if rectangle is done
        if (q.empty()) {
            //calculate area so far
            int height = abs(maxY) - abs(minY) + 1;
            int width = abs(maxX) - abs(minX) + 1;
            int area = height * width;

            if (minX == maxX && minY == maxY && !arr[minX][minY])
                area = -1;

            if(area>0)
                ans.push(area);

            minX = INT_MAX, minY = INT_MAX, maxX = -1, maxY = -1;

            //find first not visited el with value 1
            changed = false;
            for (int k = 0; k < n; k++) {
                for (int l = 0; l < n; l++) {
                    if (visited[k][l] == 0 && arr[k][l] == 1) {
                        q.push_back({ k,l });
                        visited[k][l] = 1;
                        minX = k;
                        minY = l;
                        maxX = k;
                        maxY = l;
                        changed = true;
                        break;
                    }
                    if (!visited[k][l] && !arr[k][l]) {
                        visited[k][l] = 1;
                    }
                    if (changed) {
                        break;
                    }
                }
                if (changed)
                    break;
            }
        }
    }

    while(!ans.empty()){
```

```
        printf("%d ",ans.top());
        ans.pop();
    }

    return 0;
}
```

Line: 1 Col: 1

Upload Code as File Test against custom input Run Code Submit Code