

Lonely Pixel II

Given a picture consisting of black and white pixels, and a positive integer N , find the number of black pixels located at some specific row R and column C that align with all the following rules:

1. Row R and column C both contain exactly N black pixels.
2. For all rows that have a black pixel at column C , they should be exactly the same as row R

The picture is represented by a 2D char array consisting of 'B' and 'W', which means black and white pixels respectively.

Example:

Input:

```
[['W', 'B', 'W', 'B', 'B', 'W'],
 ['W', 'B', 'W', 'B', 'B', 'W'],
 ['W', 'B', 'W', 'B', 'B', 'W'],
 ['W', 'W', 'B', 'W', 'B', 'W']]
```

$N = 3$

Output: 6

Explanation: All the bold 'B' are the black pixels we need (all 'B's at column 1 and 3).

| | 0 | 1 | 2 | 3 | 4 | 5 | column index |
|-----------|---------------------------------|---|---|---|---|---|--------------|
| 0 | ['W', 'B', 'W', 'B', 'B', 'W'], | | | | | | |
| 1 | ['W', 'B', 'W', 'B', 'B', 'W'], | | | | | | |
| 2 | ['W', 'B', 'W', 'B', 'B', 'W'], | | | | | | |
| 3 | ['W', 'W', 'B', 'W', 'B', 'W']] | | | | | | |
| row index | | | | | | | |

Take 'B' at row $R = 0$ and column $C = 1$ as an example:

Rule 1, row $R = 0$ and column $C = 1$ both have exactly $N = 3$ black pixels.

Rule 2, the rows have black pixel at column $C = 1$ are row 0, row 1 and row 2. They are exactly the same as row $R = 0$.

Note:

1. The range of width and height of the input 2D array is [1,200].

Solution 1

The difficult parts are validating the two rules:

1. Row **R** and column **C** both contain exactly **N** black pixels.
2. For all rows that have a black pixel at column **C**, they should be exactly the same as row **R**

My solution:

1. Scan each row. If that row has **N** black pixels, put a string **signature**, which is concatenation of each characters in that row, as key and how many times we see that **signature** into a HashMap. Also during scan each row, we record how many black pixels in each column in an array **colCount** and will use it in step 2.

For input:

```
[['W', 'B', 'W', 'B', 'B', 'W'],  
 ['W', 'B', 'W', 'B', 'B', 'W'],  
 ['W', 'B', 'W', 'B', 'B', 'W'],  
 ['W', 'W', 'B', 'W', 'B', 'B']]
```

We will get a HashMap:

```
{"WBWBWBW": 3, "WWBWBB": 1}
```

and colCount array:

```
[0, 3, 1, 3, 4, 1]
```

2. Go through the HashMap and if the count of one **signature** is **N**, those rows potentially contain black pixels we are looking for. Then we validate each of those columns. For each column of them has **N** black pixels (lookup in **colCount** array), we get **N** valid black pixels.

For above example, only the first **signature** "WBWBWBW" has count == 3. We validate 3 column 1, 3, 4 where char == 'B', and column 1 and 3 have 3 'B', then answer is $2 * 3 = 6$.

Time complexity analysis:

Because we only scan the matrix for one time, time complexity is $O(m*n)$. m = number of rows, n = number of columns.

```

public class Solution {
    public int findBlackPixel(char[][] picture, int N) {
        int m = picture.length;
        if (m == 0) return 0;
        int n = picture[0].length;
        if (n == 0) return 0;

        Map<String, Integer> map = new HashMap<>();
        int[] colCount = new int[n];

        for (int i = 0; i < m; i++) {
            String key = scanRow(picture, i, N, colCount);
            if (key.length() != 0) {
                map.put(key, map.getOrDefault(key, 0) + 1);
            }
        }

        int result = 0;
        for (String key : map.keySet()) {
            if (map.get(key) == N) {
                for (int j = 0; j < n; j++) {
                    if (key.charAt(j) == 'B' && colCount[j] == N) {
                        result += N;
                    }
                }
            }
        }

        return result;
    }

    private String scanRow(char[][] picture, int row, int target, int[] colCount)
    {
        int n = picture[0].length;
        int rowCount = 0;
        StringBuilder sb = new StringBuilder();

        for (int j = 0; j < n; j++) {
            if (picture[row][j] == 'B') {
                rowCount++;
                colCount[j]++;
            }
            sb.append(picture[row][j]);
        }

        if (rowCount == target) return sb.toString();
        return "";
    }
}

```

written by [shawngao](#) original link [here](#)

Solution 2

Mainly I group and count equal rows. Look for rows that appear N times and that have N black pixels. If you find one, add N for each of its black columns that doesn't have extra black pixels (in other rows).

```
def findBlackPixel(self, picture, N):
    ctr = collections.Counter(map(tuple, picture))
    cols = [col.count('B') for col in zip(*picture)]
    return sum(N * zip(row, cols).count(('B', N))
               for row, count in ctr.items()
               if count == N == row.count('B'))
```

written by [StefanPochmann](#) original link [here](#)

Solution 3

Analysis

```
/**
 * Steps:
 * >> 1. create map<int, set<int>> cols, rows; -- to store black dots on that row;
 *
 *      _0_1_2_3_4_5_
 * 0 | 0 1 0 1 1 0      rows[0] = {1, 3, 4}
 * 1 | 0 1 0 1 1 0      rows[1] = {1, 3, 4}
 * 2 | 0 1 0 1 1 0      rows[2] = {1, 3, 4}
 * 3 | 0 0 1 0 1 0      rows[3] = { 2, 4}
 *
 * >> 2. for every pixel meet rule 1, that is: pic[i][j] == 'B' && rows[i].size()
== N && cols[j].size() == N
 *      check rule2: for every row k in cols[j]; check that row[k] = row[i];
 *
 * We can tell the 6 black pixel in col 1 and col 3 are lonely pixels
 *      _0_1_2_3_4_5_
 * 0 | 0 L 0 L 1 0      rows[0] = {1, 3, 4} =
 * 1 | 0 L 0 L 1 0      rows[1] = {1, 3, 4} =
 * 2 | 0 L 0 L 1 0      rows[2] = {1, 3, 4}
 * 3 | 0 0 1 0 1 0      rows[3] = { 2, 4}
 *
 */
```

C++

```

class Solution {
public:
    int findBlackPixel(vector<vector<char>>& pic, int N) {
        int m = pic.size();
        int n = pic[0].size();
        unordered_map<int, set<int>> rows; // black pixels in each row
        unordered_map<int, set<int>> cols; // black pixels in each col
        /* 1. create map<int, set<int>> cols, rows; -- to store black dots on the
        row; */
        for (int i = 0; i < m; i++) {
            for (int j = 0; j < n; j++) {
                if (pic[i][j] == 'B') {
                    rows[i].insert(j);
                    cols[j].insert(i);
                }
            }
        }
        /* 2. for every pixel meet rule 1: pic[i][j] == 'B' && rows[i].size() == N
        && cols[j].size() == N */
        int lonelys = 0;
        for (int i = 0; i < m; i++) {
            for (int j = 0; j < n && rows.count(i); j++) {
                if (pic[i][j] == 'B' && rows[i].size() == N && cols[j].size() ==
N) { // rule 1 fulfilled
                    /* check rule2: for every row k in cols[j]; check that row[k
j] == row[i]; */
                    bool lonely = true;
                    for (int r : cols[j]) {
                        if (rows[r] != rows[i]) {
                            lonely = false; break;
                        }
                    }
                    lonelys += lonely;
                }
            }
        }
        return lonelys;
    }
};

```

Java

```

public class Solution {
    public int findBlackPixel(char[][] pic, int N) {
        int m = pic.length;
        int n = pic[0].length;
        Map<Integer, Set<Integer>> rows = new HashMap<Integer, Set<Integer>>(); /
        / black pixels in each row;
        Map<Integer, Set<Integer>> cols = new HashMap<Integer, Set<Integer>>(); /
        / black pixels in each col;
        for (int i = 0; i < m; i++) {
            for (int j = 0; j < n; j++) {
                if (pic[i][j] == 'B') {
                    if (!rows.containsKey(i)) { rows.put(i, new HashSet<Integer>(
)); }
                    if (!cols.containsKey(j)) { cols.put(j, new HashSet<Integer>(
)); }

                    rows.get(i).add(j);
                    cols.get(j).add(i);
                }
            }
        }
        int lonelys = 0;
        for (int i = 0; i < m; i++) {
            for (int j = 0; j < n && rows.containsKey(i); j++) {
                if (pic[i][j] == 'B' && rows.get(i).size() == N && cols.containsK
ey(j) && cols.get(j).size() == N) { // rule 1 fulfilled
                    int lonely = 1;
                    for (int row : cols.get(j)) {
                        if (!rows.get(i).equals(rows.get(row))) {
                            lonely = 0; break;
                        }
                    }
                    lonelys += lonely;
                }
            }
        }
        return lonelys;
    }
}

```

written by [alexander](#) original link [here](#)

From [LeetCoder](#).