Largest Piece

Problem Level: Medium

Problem Description:

It's Gary's birthday today and he has ordered his favourite square cake consisting of '0's and '1's . But Gary wants the biggest piece of '1's and no '0's . A piece of cake is defined as a part which consists of only '1's, and all '1's share an edge with each other on the cake. Given the size of cake N and the cake, can you find the count of '1's in the biggest piece of '1's for Gary ?

Sample Input 1:

```
2
1 1
0 1
```

Sample Output 1:

```
3
```

Approach to be followed:

We need to calculate the size of the largest group of 1’s that share an edge. From any index we can move in 4 directions (Top, Bottom, Right, left). Maintain a visited array to check if the element is visited or not. If it is not visited and the element is 1, mark it as visited and call recursion on all the valid directions in which we can move and increment count by 1 for every connected edge with another index at which 1 is present. Do this for until every 1 is visited and return the max count .

Steps:

1. Create a boolean visited array to store if the element is visited or not.
2. Check if the element is 1 and not visited call recursion on the helper function which will return the count.

3. Create a helper function that recursively calls over all the connected edges of 1 in all 4 directions by checking if the directional movement is valid or not and increment count if we got connected 1 edge.
4. Return the maximum count.

Pseudo Code:

```
function dfs(cake, visited, x, y, n)  //helper
    if (visited[x][y] is True)
        return 0
    visited[x][y] = true // mark as visited
    count = 1;
    dx[] = {0, 0, -1, 1}  // Change in x while taking four moves
    dy[] = {1, -1, 0, 0}  // Change in y while taking four moves

    loop from i = 0 till i is greater than equal to 4
        X = x + dx[i]
        Y = y + dy[i]
        if (X >= 0 and X < n and Y >= 0 and Y < n and cake[X][Y] is
equal to 1)
            count = count +  dfs(cake, visited, X, Y, n)
    return count



function getBiggestPieceSize(cake, n) {
    biggestPiece = 0;
    Declare visited // visited array
    Loop from i = 0 till i = n
       Loop from j = 0 till j = n
            visited[i][j] = false
    Loop from i = 0 till i is less than n
         Loop from j = 0 till j is less than n
```

```
            if (cake[i][j] equal to 1 and not visited[i][j])
                biggestPieceSize = max(biggestPieceSize, dfs(cake,
visited, i, j, n))
    return biggestPieceSize
```

Time Complexity: O(V + E), where V and E are vertices and edges of the graph formed from input.