MEDIUM

Number of Distinct Islands

Intuition

Given :

- n* m grid, we have to find distinct lands.
- 1 connected horizontally or vertically forms an island
- Two islands are considered to be distinct if and only if island is equal to another not rotated or reflected.

Eg.

1	1	0	0	0
1	1	0	0	0
0	0	0	1	1
0	0	0	1	1

Output : 1 the red islands are identical

1	1	0	0	0
1	0	0	0	0
0	0	0	1	0
0	0	0	1	1

Output : 2 as the two have same size but they are somehow rotated

We can store the shape into a set and return the length of the set this will count the unique values.

Eg.

We can check for the coordinates of the shape that we are having of the island, ie. here in the first example the shape of the first island could be :

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

1	**1**	0	0	0
1	0	0	0	0
0	0	0	**1**	**1**
0	0	0	**1**	0

We can use the list to store them as :

Island-1

```
(0,0)    (0,1)
|        |
(1,0)
```

{ (0,0), (0,1), (1,0) }

Island-1

```
(2,3)    (2,4)
|        |
(3,3)
```

{ (2,3), (2,4), (3,3) }

We will consider the source as to be the base

2,3 - 2,3 = 0,0
2,4 - 2,4 = 0,1
3,3 - 2,3 = 1,0

Now this can be used to match the first islands, so if there is a condition where the base is other than the 0,0 then we will use to subtract the base from all elements of the shape

The shape matching now could provide us with a set containing only 1 length thus this will give us output as 1

Eg.

1	**1**	0	**1**	**1**
1	0	0	0	0
0	0	0	**1**	**1**
1	**1**	0	**1**	0

[(,0,0), (0,1), (1,0) }
{ (2,3)-(2,3) = (0,0), (2,4)-(2,3) = (0,1), (3,3)-(2,3) = (!,0) } = **[(,0,0), (0,1), (1,0) }**

{ (0,3) - (0,3) = (0,0), (0,4)-(0,3) = (0,1) } = **{ (0,0), (0,1) }**
((3,0) - (3,0) = (0,0), (3,1)-(3,0) = (0,1) } = **{ (0,0), (0,1) }**

Result_Set = (**[(,0,0), (0,1), (1,0) }**, **{ (0,0), (0,1) }**)

We have to follow a same pattern for the traversal because only then we get the same ordering of the lists.

We can perform either DFS or BFS.

Eg.

```
DFS(0,0) ——-> DFS(0,1)
   | |                    |
   | |____________|
  |
DFS(1,0)
```

store(dfs(i,j,)-base)

Approach

CountDistinctIslands()

- Declare :
    - Visited vector initialized with all zeros in rows and columns of size n*m
    - Result set initially empty

- Traverse for all elements present in the grid :
    - If the element is unvisited and is 1
        - Declare an initially empty result vector to contain the DFS traversal
        - Make a DFS call as dfs(row,col,grid,visited,result,row,col)
        - Insert the element vector into the result set
- Return the length of the result set

DFS()

- Mark the node as visited
- Store the node element into the vector as node_index - source_index
- Traverse in 4 directions up, down, right, left :
    - Calculate the neighboring indexes :
        - Check if the index is not visited and grid[indexes] is 1
            - Make a dfs call as dfs(nrow,ncol,visited,grid,result,sourcerow,sourcecol)

Function Code

```
void dfs(int row,int col,vector<vector<int>> &visited,vector<vector<int>>
&grid,vector<pair<int,int>> &result,int row0,int col0)
    {
        // marking the element as visited and inserting it to the result
set as element-base
        visited[row][col] = 1;
        // Calculating element index - base
        int erow = row-row0;
        int ecol = col-col0;
        // storing into the result vector
        result.push_back({erow,ecol});
        // Computing the size of the grid

        int n = grid.size();
        int m = grid[0].size();


        // We will use a particular traversal in which we will move
particularly 4 directions

        int delRow[] = {1,0,-1,0};
        int delCol[] = {0,1,0,-1};

        for(int i=0;i<4;i++)
        {
            // calculating the neighboring elements index
            int nrow = row+delRow[i];
            int ncol = col+delCol[i];

            // checking for the validity
            if(nrow<n && ncol<m && nrow>=0 && ncol>=0)
            { // checking if its unvisited and its 1
                if(!visited[nrow][ncol] && grid[nrow][ncol]==1)
            {
                // making a dfs call for the adjacent elements
                dfs(nrow,ncol,visited,grid,result,row0,col0);
            }
            }

        }
    }
    int countDistinctIslands(vector<vector<int>>& grid) {

        // Computing the dimensions of the grid
        int n = grid.size();
        int m = grid[0].size();
        // Declaring a visited vector of same size as the grid
        vector<vector<int>> visited(n,vector<int>(m,0));

        // Creating a result_set that will contain all lists

        set<vector<pair<int,int>>> result_set;

        // Traverse for all the elements of the grid
        for(int i=0;i<n;i++)
        {


            for(int j=0;j<m;j++)
            {
                // if not visited and its a 1 then check for the DFS call
                if(!visited[i][j] && grid[i][j]==1)
                { // declaring a result vector that will contain the result
of the DFS
                  vector<pair<int,int>> result;
                  // Making a dfs call
                  dfs(i,j,visited,grid,result,i,j);
                  // inserting the result of the DFS into the set
```

```
                    result_set.insert(result);
                  }

              }


          }
          // returning the length of the set
          return result_set.size();
      }
```

Time Complexity

O(n*m*4)+O(n*m*(log(n*m))