Find the number of Islands | Set 2 (Using Disjoint Set)

Given a boolean 2D matrix, find the number of islands.

A group of connected 1s forms an island. For example, the below matrix contains 5 islands

```
{1, 1, 0, 0, 0},
{0, 1, 0, 0, 1},
{1, 0, 0, 1, 1},
{0, 0, 0, 0, 0},
{1, 0, 1, 0, 1}
```

A cell in 2D matrix can be connected to 8 neighbors.

This is an variation of the standard problem: “Counting number of connected components in a undirected graph”. We have discussed a DFS based solution in below set 1.

Find the number of islands

We can also solve the question using disjoint set data structure explained here. The idea is to consider all 1 values as individual sets. Traverse the matrix and do union of all adjacent 1 vertices. Below are detailed steps.

Approach:

1) Initialize result (count of islands) as 0
2) Traverse each index of the 2D matrix.
3) If value at that index is 1, check all its 8 neighbours. If a neighbour is also equal to 1, take union of index and its neighbour.
4) Now define an array of size row*column to store frequencies of all sets.
5) Now traverse the matrix again.
6) If value at index is 1, find its set.
7) If frequency of the set in the above array is 0, increment the result be 1.

Following is Java implementation of above steps.

```java
// Java program to fnd number of islands using Disjoint
// Set data structure.
import java.io.*;
import java.util.*;

public class Main
{
    public static void main(String[] args)throws IOException
    {
        int[][] a = new int[][] {{1, 1, 0, 0, 0},
                                 {0, 1, 0, 0, 1},
                                 {1, 0, 0, 1, 1},
                                 {0, 0, 0, 0, 0},
                                 {1, 0, 1, 0, 1}
                                };
        System.out.println("Number of Islands is: " +
                           countIslands(a));
    }

    // Returns number of islands in a[][]
    static int countIslands(int a[][])
    {
        int n = a.length;
        int m = a[0].length;

        DisjointUnionSets dus = new DisjointUnionSets(n*m);

        /* The following loop checks for its neighbours
           and unites the indexes  if both are 1. */
        for (int j=0; j<n; j++)
        {
            for (int k=0; k<m; k++)
            {
                // If cell is 0, nothing to do
                if (a[j][k] == 0)
                    continue;

                // Check all 8 neighbours and do a union
                // with neighbour's set if neighbour is
                // also 1
                if (j+1 < n && a[j+1][k]==1)
                    dus.union(j*(m)+k, (j+1)*(m)+k);
                if (j-1 >= 0 && a[j-1][k]==1)
                    dus.union(j*(m)+k, (j-1)*(m)+k);
                if (k+1 < m && a[j][k+1]==1)
                    dus.union(j*(m)+k, (j)*(m)+k+1);
                if (k-1 >= 0 && a[j][k-1]==1)
                    dus.union(j*(m)+k, (j)*(m)+k-1);
                if (j+1<n && k+1<m && a[j+1][k+1]==1)
                    dus.union(j*(m)+k, (j+1)*(m)+k+1);
                if (j+1<n && k-1>=0 && a[j+1][k-1]==1)
                    dus.union(j*m+k, (j+1)*(m)+k-1);
                if (j-1>=0 && k+1<m && a[j-1][k+1]==1)
                    dus.union(j*m+k, (j-1)*m+k+1);
                if (j-1>=0 && k-1>=0 && a[j-1][k-1]==1)
                    dus.union(j*m+k, (j-1)*m+k-1);
            }
        }

        // Array to note down frequency of each set
        int[] c = new int[n*m];
        int numberOfIslands = 0;
        for (int j=0; j<n; j++)
        {
            for (int k=0; k<m; k++)
            {
                if (a[j][k]==1)
                {

                    int x = dus.find(j*m+k);

                    // If frequency of set is 0,
                    // increment numberOfIslands
                    if (c[x]==0)
```

```
                    {
                        numberOfIslands++;
                        c[x]++;
                    }

                    else
                        c[x]++;
                }
            }
        }
        return numberOfIslands;
    }
}

// Class to represent Disjoint Set Data structure
class DisjointUnionSets
{
    int[] rank, parent;
    int n;

    public DisjointUnionSets(int n)
    {
        rank = new int[n];
        parent = new int[n];
        this.n = n;
        makeSet();
    }

    void makeSet()
    {
        // Initially, all elements are in their
        // own set.
        for (int i=0; i<n; i++)
            parent[i] = i;
    }

    // Finds the representative of the set that x
    // is an element of
    int find(int x)
    {
        if (parent[x] != x)
        {
            // if x is not the parent of itself,
            // then x is not the representative of
            // its set.
            // so we recursively call Find on its parent
            // and move i's node directly under the
            // representative of this set
            return find(parent[x]);
        }

        return x;
    }

    // Unites the set that includes x and the set
    // that includes y
    void union(int x, int y)
    {
        // Find the representatives (or the root nodes)
        // for x an y
        int xRoot = find(x);
        int yRoot = find(y);

        // Elements are in the same set, no need
        // to unite anything.
        if (xRoot == yRoot)
            return;

        // If x's rank is less than y's rank
        // Then move x under y  so that depth of tree
        // remains less
        if (rank[xRoot] < rank[yRoot])
            parent[xRoot] = yRoot;
```

```
        // Else if y's rank is less than x's rank
        // Then move y under x so that depth of tree
        // remains less
        else if(rank[yRoot]<rank[xRoot])
            parent[yRoot] = xRoot;

        else  // Else if their ranks are the same
        {
            // Then move y under x (doesn't matter
            // which one goes where)
            parent[yRoot] = xRoot;

            // And increment the the result tree's
            // rank by 1
            rank[xRoot] = rank[xRoot] + 1;
        }
    }
}
```

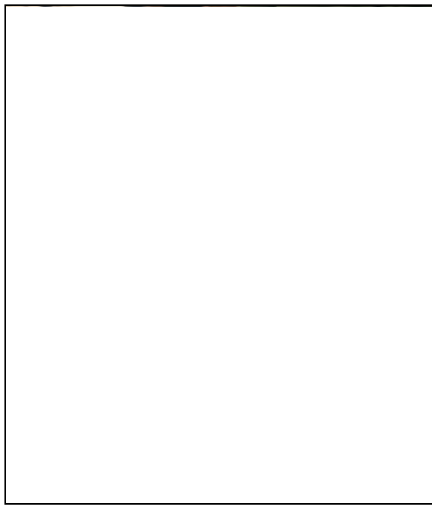
Run on IDE

Out

```
Nu
```

This by **Nikhil Tekwani** .If you like GeeksforGeeks and would like to con write an article using contribute.geeksforgeeks.org or mail your article to c eks.org. See your article appearing on the GeeksforGeeks main page and

Plea you find anything incorrect, or you want to share more information abo above.



GATE CS Corner Company Wise Coding Practice

Advanced Data Structure | Graph | graph-connectivity | union-find

Login to Improve this Article

Please write to us at contribute@geeksforgeeks.org to report any issue with the above content.

Recommended Posts:

Find the number of islands | Set 1 (Using DFS)
Disjoint Set Data Structures (Java Implementation)
Count all possible walks from a source to a destination with exactly k edges
Union-Find Algorithm | Set 2 (Union By Rank and Path Compression)
Disjoint Set (Or Union-Find) | Set 1 (Detect Cycle in an Undirected Graph)
Sorting array of strings (or words) using Trie | Set-2 (Handling Duplicates)
Union-Find Algorithm | (Union By Rank and Find by Optimized Path Compression)
Decision Tree Introduction with example
Segment Trees | (Product of given Range Modulo m)
Sparse Table

(Login to Rate)

4 Average Difficulty : **4/5.0**
Based on **37** vote(s)

Basic | Easy | Medium | Hard | Expert

Add to TODO List

Mark as DONE

Writing code in comment? Please use ide.geeksforgeeks.org, generate link and share the link here.

Load Comments | Share this post!