Find the number of islands | Set 1 (Using DFS)

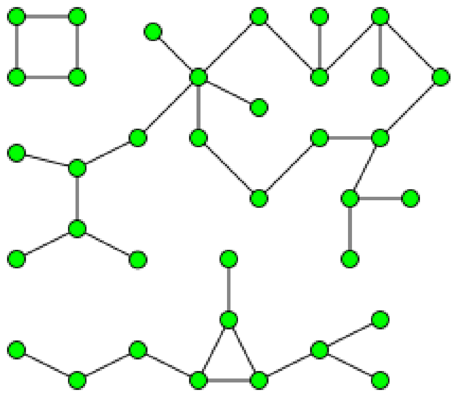
3.2

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

```
Input : mat[][] = {{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}
Output : 5
```

This is an variation of the standard problem: "Counting number of connected components in a undirected graph".

Before we go to the problem, let us understand what is a connected component. A connected component of an undirected graph is a subgraph in which every two vertices are connected to each other by a path(s), and which is connected to no other vertices outside the subgraph.
For example, the graph shown below has three connected components.



Recommended: Please solve it on "*PRACTICE*" first, before moving on to the solution.

A graph where all vertices are connected with each other, has exactly one connected component, consisting of the whole graph. Such graph with only one connected component is called as St▲y Connected Graph.

The problem can be easily solved by applying DFS() on each component. In each DFS() call, a component or a sub-graph is visited. We will call DFS on the next un-visited component. The number of calls to DFS() gives the number of connected components. BFS can also be used.

What is an island?

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}
```

Recommended: Please solve it on “*PRACTICE*” first, before moving on to the solution.

A cell in 2D matrix can be connected to 8 neighbors. So, unlike standard DFS(), where we recursively call for all adjacent vertices, here we can recursive call for 8 neighbors only. We keep track of the visited 1s so that they are not visited again.

C/C++

```
// Program to count islands in boolean 2D matrix
#include <stdio.h>
#include <string.h>
#include <stdbool.h>

#define ROW 5
#define COL 5

// A function to check if a given cell (row, col) can be included in DFS
int isSafe(int M[][COL], int row, int col, bool visited[][COL])
{
    // row number is in range, column number is in range and value is 1
    // and not yet visited
    return (row >= 0) && (row < ROW) &&
           (col >= 0) && (col < COL) &&
           (M[row][col] && !visited[row][col]);
}

// A utility function to do DFS for a 2D boolean matrix. It only considers
// the 8 neighbours as adjacent vertices
void DFS(int M[][COL], int row, int col, bool visited[][COL])
{
    // These arrays are used to get row and column numbers of 8 neighbours
    // of a given cell
    static int rowNbr[] = {-1, -1, -1,  0, 0,  1, 1, 1};
    static int colNbr[] = {-1,  0,  1, -1, 1, -1, 0, 1};

    // Mark this cell as visited
    visited[row][col] = true;

    // Recur for all connected neighbours
    for (int k = 0; k < 8; ++k)
        if (isSafe(M, row + rowNbr[k], col + colNbr[k], visited) )
            DFS(M, row + rowNbr[k], col + colNbr[k], visited);
}

// The main function that returns count of islands in a given boolean
// 2D matrix
int countIslands(int M[][COL])
{
    // Make a bool array to mark visited cells.
    // Initially all cells are unvisited
    bool visited[ROW][COL];
    memset(visited, 0, sizeof(visited));

    // Initialize count as 0 and travese through the all cells of
    // given matrix
    int count = 0;
    for (int i = 0; i < ROW; ++i)
        for (int j = 0; j < COL; ++j)
            if (M[i][j] && !visited[i][j]) // If a cell with value 1 is not
            {                              // visited yet, then new island found
                DFS(M, i, j, visited);     // Visit all cells in this island.
                ++count;                   // and increment island count
            }

    return count;
}

// Driver program to test above function
int main()
{
    int M[][COL]= {  {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}
    };

    printf("Number of islands is: %d\n", countIslands(M));

    return 0;
}
```

Run on IDE

Java

```
// Java program to count islands in boolean 2D matrix
import java.util.*;
import java.lang.*;
import java.io.*;

class Islands
{
    //No of rows and columns
    static final int ROW = 5, COL = 5;

    // A function to check if a given cell (row, col) can
    // be included in DFS
    boolean isSafe(int M[][], int row, int col,
                   boolean visited[][])
    {
        // row number is in range, column number is in range
        // and value is 1 and not yet visited
        return (row >= 0) && (row < ROW) &&
               (col >= 0) && (col < COL) &&
               (M[row][col]==1 && !visited[row][col]);
    }

    // A utility function to do DFS for a 2D boolean matrix.
    // It only considers the 8 neighbors as adjacent vertices
    void DFS(int M[][], int row, int col, boolean visited[][])
    {
        // These arrays are used to get row and column numbers
        // of 8 neighbors of a given cell
        int rowNbr[] = new int[] {-1, -1, -1,  0, 0,  1, 1, 1};
        int colNbr[] = new int[] {-1,  0,  1, -1, 1, -1, 0, 1};

        // Mark this cell as visited
        visited[row][col] = true;

        // Recur for all connected neighbours
        for (int k = 0; k < 8; ++k)
            if (isSafe(M, row + rowNbr[k], col + colNbr[k], visited) )
                DFS(M, row + rowNbr[k], col + colNbr[k], visited);
    }

    // The main function that returns count of islands in a given
    //  boolean 2D matrix
    int countIslands(int M[][])
    {
        // Make a bool array to mark visited cells.
        // Initially all cells are unvisited
        boolean visited[][] = new boolean[ROW][COL];


        // Initialize count as 0 and travese through the all cells
        // of given matrix
        int count = 0;
        for (int i = 0; i < ROW; ++i)
            for (int j = 0; j < COL; ++j)
                if (M[i][j]==1 && !visited[i][j]) // If a cell with
                {                                 // value 1 is not
                    // visited yet, then new island found, Visit all
                    // cells in this island and increment island count
                    DFS(M, i, j, visited);
                    ++count;
                }

        return count;
    }

    // Driver method
    public static void main (String[] args) throws java.lang.Exception
    {
        int M[][]=  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}
                                };
        Islands I = new Islands();
        System.out.println("Number of islands is: "+ I.countIslands(M));
    }
} //Contributed by Aakash Hasija
```

Run on IDE

Python

```
# Program to count islands in boolean 2D matrix
class Graph:

    def __init__(self, row, col, g):
        self.ROW = row
        self.COL = col
        self.graph = g

    # A function to check if a given cell
    # (row, col) can be included in DFS
    def isSafe(self, i, j, visited):
        # row number is in range, column number
        # is in range and value is 1
        # and not yet visited
        return (i >= 0 and i < self.ROW and
                j >= 0 and j < self.COL and
                not visited[i][j] and self.graph[i][j])


    # A utility function to do DFS for a 2D
    # boolean matrix. It only considers
    # the 8 neighbours as adjacent vertices
    def DFS(self, i, j, visited):

        # These arrays are used to get row and
        # column numbers of 8 neighbours
        # of a given cell
        rowNbr = [-1, -1, -1,  0, 0,  1, 1, 1];
        colNbr = [-1,  0,  1, -1, 1, -1, 0, 1];

        # Mark this cell as visited
        visited[i][j] = True

        # Recur for all connected neighbours
        for k in range(8):
            if self.isSafe(i + rowNbr[k], j + colNbr[k], visited):
                self.DFS(i + rowNbr[k], j + colNbr[k], visited)


    # The main function that returns
    # count of islands in a given boolean
    # 2D matrix
    def countIslands(self):
        # Make a bool array to mark visited cells.
        # Initially all cells are unvisited
        visited = [[False for j in range(self.COL)]for i in range(self.ROW)]

        # Initialize count as 0 and travese
        # through the all cells of
        # given matrix
        count = 0
        for i in range(self.ROW):
            for j in range(self.COL):
                # If a cell with value 1 is not visited yet,
                # then new island found
                if visited[i][j] == False and self.graph[i][j] ==1:
                    # Visit all cells in this island
                    # and increment island count
                    self.DFS(i, j, visited)
                    count += 1

        return count


graph = [[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]]


row = len(graph)
col = len(graph[0])

g= Graph(row, col, graph)

print "Number of islands is :"
print g.countIslands()

#This code is contributed by Neelam Yadav
```

Run on IDE

Output:

```
Number of islands is: 5
```

Time complexity: O(ROW x COL)

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

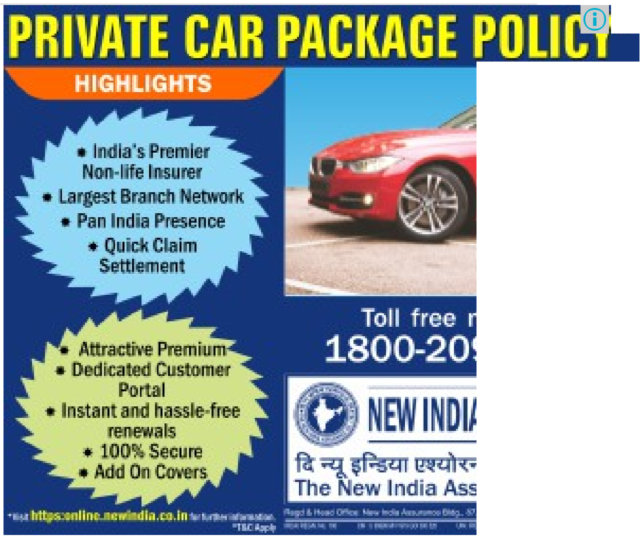
Asked in: Amazon, Citrix, DE Shaw, Informatica, Microsoft, Ola, One97, Opera, Paytm, Snapdeal, Streamoid Technologi, Visa

Reference:
http://en.wikipedia.org/wiki/Connected_component_%28graph_theory%29

Please write comments if you find anything incorrect, or you want to share more information abc topic discussed above.

.COM Domain at Just ₹99

Secure The Domain Name You Want Before it's Taken Buy Now!



GATE CS Corner   Company Wise Coding Practice

Graph Matrix DFS graph-connectivity   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 2 (Using Disjoint Set)
Depth First Traversal or DFS for a Graph
Find length of the largest region in Boolean Matrix
Count all possible walks from a source to a destination with exactly k edges
Count number of islands where every island is row-wise and column-wise separated
Minimum initial vertices to traverse whole matrix with given conditions
Shortest path to reach one prime to other by changing single digit at a time
Traveling Salesman Problem (TSP) Implementation
Degree Centrality (Centrality Measure)
Water Jug problem using BFS

▲

(Login to Rate)

3.2 Average Difficulty : **3.2/5.0**
Based on **147** 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!

▲