Graphs

Content

Applications

1. 3 Cycle

Problem

Given a graph with N vertices (numbered from 0 to N-1) and M undirected edges, then count the distinct 3-cycles in the graph. A 3-cycle PQR is a cycle in which (P,Q), (Q,R) and (R,P) are connected by an edge.

Approach

1. We will go to each vertex & check if there is any 3 cycle that can be formed & increment count if there is a 3-cycle.
2. We can solve this iteratively.

3. Go to each vertex, then go to each adjacent vertex one by one.
4. Then go to the adjacent edges of the adjacent vertex of startingVertex & ignore the startingVertex in the adjacent edges.
5. Check if those adjacent edges -> vertices also have edges with the startingVertex.
6. If yes, count++, else move to other vertices.

Implementation

https://github.com/lavishabhambri/Weekly-Algo-Newsletter/blob/main/Graphs/Codes/3Cycle.cpp

2. Minimum Number of swaps to sort an array

Problem

We are given an array of n distinct elements and are required to find the minimum number of swaps required to sort the array.

Approach

We understand this problem for the following example.

In figure 1, we are given an unsorted 1D array. Their current positions are written above them. We sort the given array to find out the sorted position of each element.

Before sorting, a new 1D array is made in which every element is inserted along with their unsorted positions as shown in figure 2.

Now, this array is sorted and looks like figure 3.

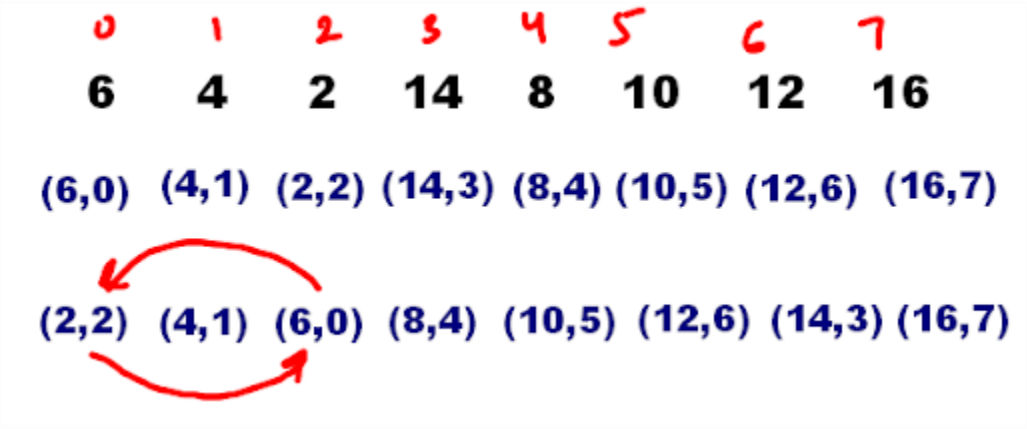
The elements like (4,1) and (16,7) which are the same position before and after the sorting, remain untouched.

Instead of calculating the minimum number of swaps to sort the unsorted array, we count the minimum number of swaps to bring the sorted array back to its original unsorted form.

So, for the remaining elements in the sorted array, we try to put the elements in place of those elements which are present at their previous unsorted positions to count the minimum swaps required to achieve the sorted array.
Hence, we retrace our steps.

For example, to retrace our steps we move element 6 from 2nd position, back to its original 0th position.

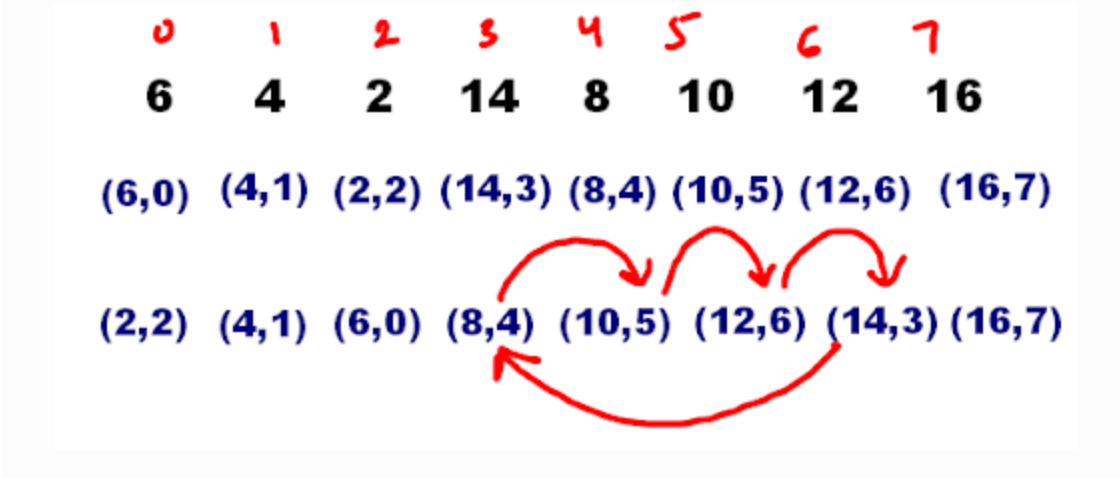
To make room for element 6, the element 2 at the 0th position is moved to its original position, which turns out to be index 2.



Hence, a cycle seems to have formed and in this cycle, there are 2 elements where the total number of swaps is 1.

Hence, we see that the elements in a cycle get swapped with each other. Let's prove this statement for some other cycle.

In the sorted array, element 8 is at 3rd position. But it came from the 4th position. So let's put it back there.
For performing this process, we need to put the element 10 which is currently at the 4th position back to its unsorted position of 5th index.
Again, for doing this, the element at the 5th index in the sorted array should be placed at its original 6th index where currently element 14 is residing.
This element 14 is sent back to its original 3rd position.



Reader, meditate on the fact that we started backtracking our steps from the 3rd position and have ended at the same 3rd position, hence forming a cycle. Therefore, we see that the elements in a cycle get swapped with each other.
In this cycle we see that for 4 elements, the number of swapping is 3.

Hence, from the above two cycles, we derive that the number of swapping is 1 less than the total elements in a cycle.

So, our approach to this problem should be to find the total number of cycles. Then find the number of elements in those cycles. The number of swaps is 1 less than the number of elements in each cycle. The total number of swaps from all the cycles is then returned.

Implementation

https://github.com/lavishabhambri/Weekly-Algo-Newsletter/blob/main/Graphs/Codes/minimumNumberOfSwapsToSortAnArray.cpp

3. Number of Distinct islands

Problem

Given an m*n binary matrix mat, return the number of distinct island.

An island is considered to be the same as another if and only if one island can be translated (and not rotated or reflected) to equal the other.

Approach

The core of the question is to know if 2 islands are equal. The primary criteria is that the number of 1's should be same in both. But this cannot be the only criteria as we have seen in example 2 above. So how do we know? We could use the position/coordinates of the 1's.

If we take the first coordinates of any island as a base point and then compute the coordinates of other points from the base point, we can eliminate duplicates to get the distinct count of islands. So, using this approach, the coordinates for the 2 islands in example 1 above can be represented as: [(0, 0), (0, 1), (1, 0), (1, 1)].

Implementation

https://github.com/lavishabhambri/Weekly-Algo-Newsletter/blob/main/Graphs/Codes/number_of_distinct_islands.cpp

4. Number of enclaves

Problem

You are given an m x n binary matrix grid, where 0 represents a sea cell and 1 represents a land cell.

A move consists of walking from one land cell to another adjacent (4-directionally) land cell or walking off the boundary of the grid.

Return the number of land cells in grid for which we cannot walk off the boundary of the grid in any number of moves.

Approach

The answer will be 3, as there are three ones enclosed by 0s, and one 1 is not enclosed.

To solve this, we will follow these steps −

- Make a direction array dir, and store [[1,0], [-1,0], [0,1], [0,-1]]
- make a method called dfs() this will take x, y and the matrix A
- if x < 0 or y > 0 or x > row count of A or y > column counts of A, or A[x, y] is 0, then return
- set A[x, y] := 0
- for k in range 0 to 3
    - nx := dir[k, 0] + x, ny := dir[k, 1] + y
    - dfs(nx, ny, A)
- From the main method do the following
- ret := 0, n := row count of A
- m := column count of A if n is not 0, otherwise m := 0
- for i in range 0 to n – 1
    - if A[i, 0] = 1, then call dfs(i, 0, A)
    - if A[i, m – 1] is 1, then call dfs(i, m – 1, A)
- for i in range 0 to m – 1
    - if A[0, i] = 1, then call dfs(0, i, A)
    - if A[n – 1, i] is 1, then call dfs(n – 1, i, A)
- for i in range 0 to n – 1
    - for j in range 0 to m – 1
        - ret := ret + A[i, j]
- return ret

Implementation

https://github.com/lavishabhambri/Weekly-Algo-Newsletter/blob/main/Graphs/Codes/number_of_enclaves.cpp