Assignment Solutions : Graph - 3 Problems on graph Coloring

Q1 You are given a list of bombs. The range of a bomb is defined as the area where its effect can be felt. This area is in the shape of a circle with the center as the location of the bomb. The bombs are represented by a 0-indexed 2D integer array bombs where bombs[i] = [xi, yi, ri]. xi and yi denote the X-coordinate and Y-coordinate of the location of the ith bomb, whereas ri denotes the radius of its range.

You may choose to detonate a single bomb. When a bomb is detonated, it will detonate all bombs that lie in its range. These bombs will further detonate the bombs that lie in their ranges.

Given the list of bombs, return *the maximum number of bombs that can be detonated if you are allowed to detonate only one bomb.*

Code link: https://pastebin.com/zTtkSAgx

Explanation:

- **We define a struct Bomb to represent each bomb, containing its x-coordinate (x), y-coordinate (y), and radius (r).**
- **The function maxDetonatedBombs takes a vector of Bomb objects as input and returns the maximum number of bombs that can be detonated.**
- **We initialize a variable maxDetonated to keep track of the maximum number of detonated bombs. We also store the total number of bombs in a variable n.**
- **We iterate through each bomb in the input vector using a for loop. For each bomb at index i, we perform a Breadth-First Search (BFS) starting from that bomb to find the number of bombs that can be detonated.**
- **Inside the BFS loop, we initialize a visited array of size n to keep track of which bombs have been visited during the BFS. We mark the current bomb (i) as visited.**
- **We use a queue (q) to store the indices of the bombs that need to be visited. We start by enqueueing the index i.**
- **While the queue is not empty, we dequeue the front element (curr) and increment the detonated count.**

- **We iterate through all the bombs (j) and check if the bomb at index j has not been visited yet. If it hasn't, we calculate the square of the Euclidean distance between the current bomb (curr) and the bomb at index j using their coordinates. We compare this distance with the square of the range of the current bomb (r).**
- **If the distance is within the range of the current bomb, we mark the bomb at index j as visited, enqueue its index (j) to the queue, and continue with the next iteration of the loop.**
- **Once the BFS loop completes, we have found the number of bombs that can be detonated starting from the current bomb (i). We update the maxDetonated count if the detonated count is greater than the current maximum.**
- **Finally, after the outer for loop finishes iterating through all the bombs, we return the maximum number of detonated bombs (maxDetonated).**

The BFS approach allows us to explore the bombs in a breadth-first manner, ensuring that we count the maximum number of bombs that can be detonated by considering the bombs that are within the range of each other. By marking visited bombs and using a queue to manage the order of exploration, we systematically cover all possible detonation scenarios.

Output:

```
Maximum number of bombs that can be detonated: 2


...Program finished with exit code 0
Press ENTER to exit console.
```

Q2 You are given two m x n binary matrices grid1 and grid2 containing only 0's (representing water) and 1's (representing land). An island is a group of 1's connected 4-directionally (horizontal or vertical). Any cells outside of the grid are considered water cells.
An island in grid2 is considered a sub-island if there is an island in grid1 that contains all the cells that make up this island in grid2.
Return the number of islands in grid2 that are considered sub-islands.

Code link: https://pastebin.com/1AZUSTTS

Explanation:

The provided code also solves the problem using a Depth-First Search (DFS) approach. Here's an explanation of the approach used in the code:

- **The function dfs is a helper function that performs a Depth-First Search on grid2. It marks the visited land cells in grid2 with a value of 2. It also checks if the corresponding cell in grid1 is land (value of 1) and sets a flag to 0 if it's not.**
- **The function countSubIslands takes two grid matrices, grid1 and grid2, as input.**
- **Initialize a variable ans to keep track of the number of sub-islands found.**
- **Iterate through each cell of grid2 using nested loops. If a land cell is found (with a value of 1) and it matches the corresponding cell in grid1 (also a value of 1), perform the following steps:**
  - **Initialize a flag variable as 1.**
  - **Call the dfs function to perform a DFS on the land cell in grid2 and check if it is a sub-island. The flag variable is passed by reference to update it within the dfs function.**
  - **Increment ans by the value of the flag (1 if it is a sub-island, 0 otherwise).**
- **Return the ans variable.**

In the provided example, the code determines that there are 2 sub-islands in grid2. The output will be "Number of sub-islands in grid2: 2".

Output:

```
Number of sub-islands in grid2: 3


...Program finished with exit code 0
Press ENTER to exit console.
```

Q3 You are given a 0-indexed m x n binary matrix land where a 0 represents a hectare of forested land and a 1 represents a hectare of farmland.
To keep the land organized, there are designated rectangular areas of hectares that consist entirely of farmland. These rectangular areas are called groups. No two groups are adjacent, meaning farmland in one group is not four-directionally adjacent to another farmland in a different group.
land can be represented by a coordinate system where the top left corner of land is (0, 0) and the bottom right corner of land is (m-1, n-1). Find the coordinates of the top left and bottom right corner of each group of farmland. A group of farmland with a top left corner at (r1, c1) and a bottom right corner at (r2, c2) is represented by the 4-length array [r1, c1, r2, c2].
Return a 2D array containing the 4-length arrays described above for each group of farmland in land. If there are no groups of farmland, return an empty array. You may return the answer in any order.

Code link : https://pastebin.com/5LDc7YjA

Explanation:

- **The nbrs vector represents the four neighboring directions: right, down, up, and left. Each direction is represented by a pair of integers that represents the change in row (nbr[0]) and column (nbr[1]) coordinates.**
- **The dfs function performs a Depth-First Search on the land matrix. It takes the land matrix, current row and column coordinates (i, j), and a visited matrix as input. The visited matrix is used to keep track of the visited cells during the DFS.**
- **Inside the dfs function:**
  - **The current cell at (i, j) is marked as visited by setting the corresponding element in the visited matrix to true.**
  - **A pair of integers res is initialized with the current row and column coordinates (i, j).**
  - **For each neighboring cell, the function checks if it is within the matrix boundaries, is not visited, and is a farmland cell (with a value of 1).**
  - **If these conditions are met, the dfs function is called recursively on the neighboring cell, and the resulting pair of integers ans is obtained.**
  - **The res pair of integers is updated by taking the maximum row and column coordinates between the current cell and the neighboring cell.**
  - **Finally, the res pair is returned.**
- **The findFarmland function takes the land matrix as input and returns a 2D vector containing the top left and bottom right coordinates of each group of farmland.**
- **Inside the findFarmland function:**
  - **The dimensions of the land matrix (m rows and n columns) are obtained.**
  - **A visited matrix of size m x n is created and initialized with false values.**
  - **An empty 2D vector ans is created to store the results.**
  - **The code iterates through each cell of the land matrix using nested loops.**
  - **If a cell is not visited and is a farmland cell (with a value of 1), the dfs function is called on that cell, and the resulting pair of integers p is obtained.**
  - **The top left and bottom right coordinates of the group of farmland are stored in a vector res.**
  - **The res vector is added to the ans vector.**
  - **Finally, the ans vector is returned.**
- **In the main function, the findFarmland function is called with the provided land matrix. The results are printed in the same format as before.**

The code performs a DFS to find the groups of farmland in the land matrix. It utilizes a recursive approach and keeps track of visited cells. For each unvisited farmland cell encountered, it explores all the adjacent farmland cells to find the top left and bottom right coordinates of the group.

Output:

```
1
1
Groups of farmland:
[0, 0, 0, 0]
[1, 1, 2, 2]


...Program finished with exit code 0
Press ENTER to exit console.
```

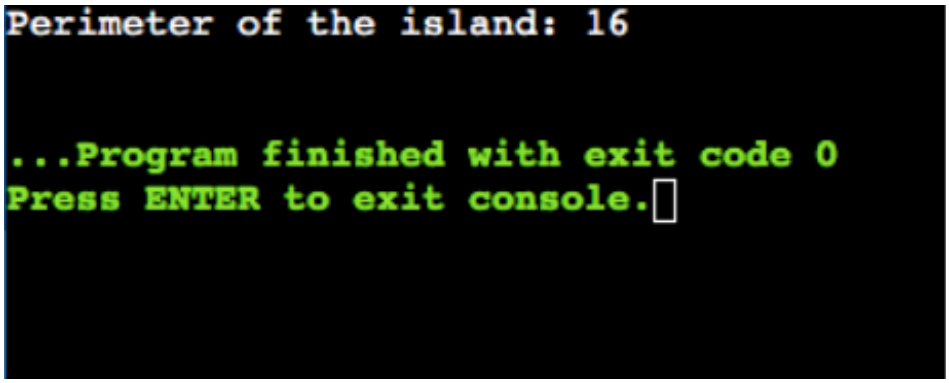
Q4 You are given row x col grid representing a map where grid[i][j] = 1 represents land and grid[i][j] = 0 represents water.
Grid cells are connected horizontally/vertically (not diagonally). The grid is completely surrounded by water, and there is exactly one island (i.e., one or more connected land cells).
The island doesn't have "lakes", meaning the water inside isn't connected to the water around the island. One cell is a square with side length 1. The grid is rectangular, width and height don't exceed 100. Determine the perimeter of the island.

Code link : https://pastebin.com/bb5cvb4t

Explanation:

- **The islandPerimeter function takes a 2D vector grid representing the land and water cells. It returns the perimeter of the island.**
- **Initialize the perimeter variable to 0 to keep track of the total perimeter.**
- **Get the number of rows (row) and columns (col) in the grid.**
- **Iterate through each cell of the grid using nested loops.**
- **If a land cell is encountered (with a value of 1), increment the perimeter by 4, as each land cell contributes 4 sides to the perimeter.**
- **Check if the cell has a neighboring land cell to the left (grid[i][j-1]) and above (grid[i-1][j]). If so, decrement the perimeter by 2, as the shared sides between two adjacent land cells should not be counted in the perimeter.**
- **After processing all the cells, return the perimeter.**

Output:


Perimeter of the island: 16
...Program finished with exit code 0
Press ENTER to exit console.