Algorithms: Design and Analysis - CS 412

Weekly Challenge 04

Ali Muhammad Asad - aa07190

1. (1 point) Imagine that you have a 2D grid where each cell contains a color value (see Figure 1 for an example). Given a start location (x, y), target color, and replacement color, design a recursive algorithm to fill closed regions with a specific color (replacement color). Clearly define a base criterion and analyze the worst-case time complexity of your proposed algorithm/approach.

Suppose that you have the following array:

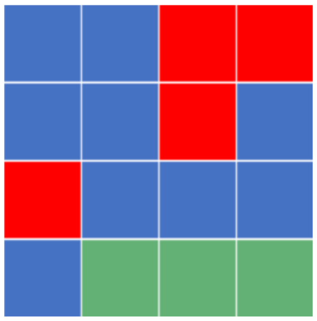

Figure 1: Before

Now, we're given a $start\_position = (0, 0)$ and $new\_color =$ orange. As we can see the color of cell $(0, 0)$ is blue. Therefore, we'll recolor all its connected cells that have the same color to the new_color:

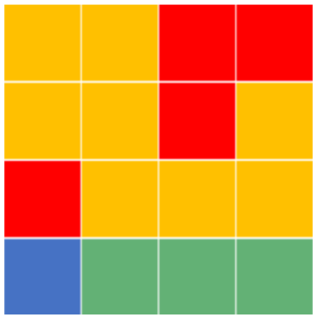

Figure 2: After

Input: start location (row, column), replacement color.
Output: replaces all adjacent cells with the target color with the replacement color. The target color is the original color of the starting location.
Moves: possible moves are up, down, left, right. In addition to up, down, left, and right, you can also add up-right, up-left, down-right, and down-left.
Hint: The algorithm might include recursive calls for each move.

Note: Figure 1 is only an example, your algorithm should be valid for any number of colors and any arrangement of the colors.

Solution: We are given a 2D array, where each cell contains a color value. Then our 2D array can be of size $n \times m$, where n is the rows (number of sub-arrays in our array), and m are the columns (number of elements in each sub-array). Then we need to design an algorithm that replaces the colour of the starting cell with the new color, along with cells sharing the same color that are connected to it. Then for any location (i, j), or (x, y) we check the surrounding points $(x+1, y), (x-1, y), (x, y+1), (x, y-1)$ if they have the same color as the starting location, and replace its color.

This can also be thought of as a graph, where each node has some edges connecting to some neighbors, and we have to explore as far as possible before backtracking (in the recursive-based approach), filling the connected nodes with the new color. Then a Depth-First Search (DFS) inspired / based approach can be implemented that works as follows:

1. Start at the given location
2. Check if the starting row (sr) and starting column (sc) are within the grid boundaries (row and col) to avoid going out of bounds.
3. Check if the color at the starting position matches the target color. If not, the area doesn't belong to the region, so return.
4. Fill and Explore:
   (a) Change the color of the current cell to the new color.
   (b) Recursively explore neighbors by checking all four adjacent neighbors; left, right, up, down, using the new start point as $(x+1, y), (x-1, y), (x, y+1), (x, y-1)$.
   (c) For each neighbor, repeat steps 1-3, ensuring that only connected regions are filled.

A pseudocode for the above given code is as follows:

```
function fillColor(grid, s_row, s_col, row, col, source, color):
  // Check if the given location is within the grid boundaries
  if s_row < 0 or s_row >= row or s_col < 0 or s_col >= col:
    return

  // Check if the color at the position matches the target color
  if grid[s_row][s_col] != source:
    return

  // Mark the cell with the new color
  grid[s_row][s_col] = color

  // Recursively explore the surrounding neighbors
  fillColor(grid, s_row + 1, s_col, row, col, source, color)
  fillColor(grid, s_row - 1, s_col, row, col, source, color)
  fillColor(grid, s_row, s_col + 1, row, col, source, color)
  fillColor(grid, s_row, s_col - 1, row, col, source, color)
end function
```

The worst-case time complexity of the above algorithm is the same as the DFS Complexity which is $O(n \times m)$, where n is the number of rows, and m is the number of columns. This is because in the worst case, the algorithm may need to visit each and every cell in the grid, which can happen if the whole grid is filled with a single color. Since we designed our algorithm to only return if the cell does not match the target color, then each cell is only visited once. Hence, the worst time complexity is $O(n \times m)$.