# Analysis of Algorithms 2

## BLG 336E

# Project 1 Report

Kerem Er
erk21@itu.edu.tr

Faculty of Computer and Informatics Engineering
Department of Computer Engineering
Date of submission: 24.03.2024

## 0.1. Introduction

In this project, I implemented BFS and DFS algorithms to use in finding top k colonies.

## 0.2. Pseudocodes

### 0.2.1. DFS

This DFS code is depth-first searching on a 2D map to calculate the size of a contiguous area of same resource values. It navigates through the grid recursively, wrapping around edges if necessary, and sums up cells with the same resource. The time complexity is O(m*n).

```
DFS(map, row, col, resource):
    wrapped = getWrappedIndices(row, col, map)
    row = wrapped.first
    col = wrapped.second

    if visited[row][col] or map[row][col]  resource:
        return 0
    visited[row][col] = true

    size = 1
    size += DFS(map, row - 1, col, resource)
    size += DFS(map, row + 1, col, resource)
    size += DFS(map, row, col - 1, resource)
    size += DFS(map, row, col + 1, resource)

    return size
```

### 0.2.2. BFS

BFS function counts cells with the same value in the map, using a queue to visit each cell only once. It returns the number of cells connected to the starting cell that have the matching value. The time complexity is O(m*n).

```
BFS(map, row, col, resource):
    initVisited(map)
    wrapped = getWrappedIndices(row, col, map)

    if visited[wrapped.first][wrapped.second]
    or map[wrapped.first][wrapped.second]  resource:
    return 0
    q.push(wrapped)
    visited[wrapped.first][wrapped.second] = true
    size = 0
    while q is not empty:
        front = q.front(), q.pop()
        size++

        neighbors = {
        getWrappedIndices(front.first - 1, front.second, map),
        getWrappedIndices(front.first + 1, front.second, map),
        getWrappedIndices(front.first, front.second - 1, map),
        getWrappedIndices(front.first, front.second + 1, map) }

        for neighbor in neighbors:
            nr = neighbor.first
            nc = neighbor.second
            if map[nr][nc] is not visited and map[nr][nc] = resource:
                visited[nr][nc] = true
                q.push(neighbor)

    return size
```

### 0.2.3. Top K Largest Colonies

This function finds the top k of the largest resource colonies in a grid `map` using either DFS or BFS, based on the `useDFS` flag. It returns a list of these colonies, each represented by its size and type. The function also measures and outputs the execution time. The time complexity is O(m*n + c log c).

```
top_k_largest_colonies(map, useDFS, k):
    start = currentTime
    if map or map[0] is empty:
        return {}

    initVisited(map)
    for i in mapSize:
        for j in map[i]Size:
            if map[i][j] is not visited and bigger than 0:
                resourceType = map[i][j]
                if useDFS is true:
                    dfs(map, i, j, resourceType)
                else:
```

```
                bfs(map, i, j, resourceType)
            colonySizes.emplace_back(size, resourceType)

sort(colonySizes.begin(), colonySizes.end(), colonyComparator)
if colonySizes size is bigger than k:
    colonySizes.resize(k)

stop = currentTime
duration = stop - start
print duration

return colonySizes
```

## 0.3. Questions

### 0.3.1. Q1 - Why should you maintain a list of discovered nodes? How does this affect the outcome of the algorithms?

- Avoiding Infinite Loops
  - Without a list to track visited nodes, both DFS and BFS could keep revisiting the same nodes indefinitely .
- Preventing Redundancy
  -Tracking which nodes have been discovered ensures that each node is processed only once.
- To count correctly
  - Without maintaining a list to track the nodes, we can not count correctly.

### 0.3.2. Q2 - How does the map size affect the performance of the algorithm for finding the largest colony in terms of time and space complexity

- Time Complexity

  -If the map is represented as a 2D grid of size `m x n`, the complexity will be O(m*n). So, the largest map is the complicated time is.
- Space Complexity

  -For both DFS and BFS, space complexity involves maintaining a 2D visited array of the same dimensions as the map, leading to a space complexity of O(m*n), matching the map's size.

### 0.3.3. Q3 - How does the choice between Depth-First Search (DFS) and Breadth-First Search (BFS) affect the performance of finding the largest colony?

Yes, it affects the performance according to various things. DFS might be more efficient in terms of memory usage in certain cases, however BFS has a more consistent exploration pattern that could be better depending on the colonies' layout. So, the choice depends on various things.