# ICS202 Lab Project Report

## Introduction

This report documents the implementation of the lab project for ICS202. The project involves working with the quadtree data structure to handle image processing tasks. A quadtree is a hierarchical tree structure that recursively partitions a 2-D space into four quadrants.

The project focuses on creating a Java class, ImgQuadTree, to represent and manipulate a quadtree for a 256 x 256 grayscale image. The tasks include constructing the quadtree from a preorder traversal file, calculating the number of nodes and leaves, and converting the quadtree back into an image array. Additionally, I developed another class, ImgQuadTreeFileCreator, to convert uncompressed image data into a quadtree representation. The final goal is to demonstrate the effectiveness of the quadtree in reducing the image size while preserving the image's details.

## Problem-Solving Strategy for Task A

### 1. Reading the Quadtree File

- **Objective**: Read the walkingtotheskyQT.txt file and build the quadtree from it.

- **Approach**:

- Implement a constructor ImgQuadTree(String filename) that opens the file using a Scanner.

- If the file cannot be opened, print an error message and exit.

- Use the buildTree(Scanner reader) method to construct the quadtree from the file. This method reads integers from the file and recursively builds the tree.

### 2. Building the Quadtree

- **Objective**: Construct the quadtree based on integers read from the file.

- **Approach**:

- The buildTree(Scanner reader) method reads the next integer:

- If the integer is -1, it indicates a non-leaf node. Recursively create four children nodes.

- If the integer is between 0 and 255, it indicates a leaf node with that intensity value, create the node and return it, for the previous call to finish collecting the four children, which will eventually construct the whole tree.

### 3. Counting the Number of Nodes

- **Objective**: Count the total number of nodes in the quadtree.

- **Approach**:

- Implement the getNumNodes() method, which calls getNumNodesHelper(QTNode current).

- The helper method traverses the tree recursively, adding `1` for each node visited.

- Base case: If the current node is null, return 0.

### 4. Counting the Number of Leaves

- **Objective**: Count the total number of leaf nodes in the quadtree.

- **Approach**:

- Implement the getNumLeaves() method, which calls getNumLeavesHelper(QTNode current).

- The helper method traverses the tree recursively:

- If the current node is a leaf (intensity level between `0 and 255`), add 1.

- if the current node is null i.e. we reached the end, return 0.

- If the current node is `-1`, call the function on each of the children and add them up.

### 5. Generating the Image Array

- **Objective**: Convert the quadtree back into a 256 x 256 image array.

- **Approach**:

- Implement the getImageArray() method, which calls getImageArrayHelper(int[][] arr, QTNode current, int length).

- The helper method recursively builds the image array:

- Base case: If the length is less than 1, then work is finished; return the array.

- If the current node is a non-leaf (-1), recursively create four smaller arrays representing the four quadrants and call the function on these quarters with the corresponding node, which will either do the same again or fill it with a certain intensity value.

- Combine these quadrants into the larger array using the copy2DArray(int[][] src, int[][] dest, int startRow, int startCol) method.

- If the current node is a leaf, fill the array with the leaf's intensity value and return it.

### 6. Helper Methods/Classes

- **Objective**: Assist in array manipulation and node creation.

- **Approach**:

- copy2DArray(int[][] src, int[][] dest, int startRow, int startCol): Copy a smaller array into a specific quadrant of a larger array.

- QTNode inner class: Represents nodes in the quadtree, with constructors for leaf and non-leaf nodes.

## Screenshots for Task A

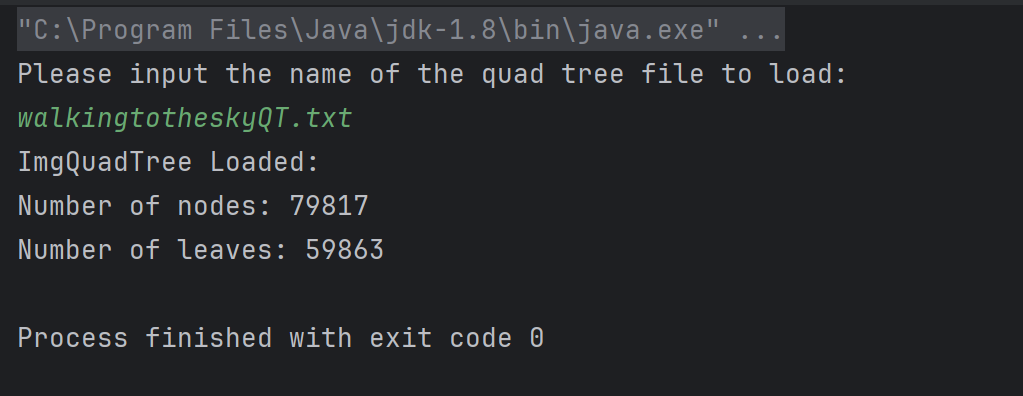


Figure 1: Terminal Output

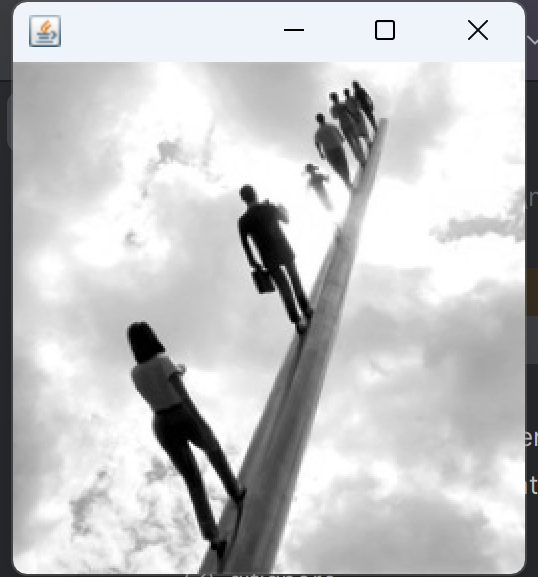


Figure 2: Resulting Image for walkingtotheskyQT.txt

## Problem-Solving Strategy for Task C

### 1. Reading the Uncompressed Image File

- **Objective**: Read the smilyface.txt file and convert it into a 256 x 256 image array.

- **Approach**:

- Use the constructor ImgQuadTreeFileCreator() to prompt the user for the filename.

- Open the file using a Scanner and handle FileNotFoundException.

- Call getImageArray(Scanner reader, int length) to create a 2D array from the file data.

### 2. Creating the Image Array

- **Objective**: Convert the file's integer data into a 256 x 256 image array.

- **Approach**:

- Implement the getImageArray(Scanner reader, int length) method.

- Read each integer from the file and fill the array row by row, meaning the 1st 256 integers represent the 1st row, the second represents the 2nd row, and so on.

### 3. Generating the Preorder Traversal

- **Objective**: Create a preorder traversal list from the image array that represents the quadtree.

- **Approach**:

- Implement getPreOrderTraversal(int[][] imgArray) to recursively generate the preorder traversal.

- Base case: If the array contains only one unique value, return a list with that value.

- Recursive case: If the array contains multiple values:

- Add -1 to the list to indicate a non-leaf node.

- Divide the array into four quadrants and recursively process each quadrant.

- Combine the results to form the complete preorder traversal list.

### 4. Writing the Preorder Traversal to a File

- **Objective**: Write the preorder traversal list to a new quadtree file one integer per line.

- **Approach**:

- Implement writeTraversal(ArrayList<Integer> preOrderTraversal, String filename).

- Create a PrintWriter, loop on each integer from the traversal list, and write to the file with PrintWriter.println().

- Handle FileNotFoundException.

### 5. Helper Methods

- **Objective**: Assist in checking array values and extracting quadrants.

- **Approach**:

- isOneValue(int[][] arr): Check if the entire array contains only one unique value.

- Loop through the array and compare each element to the first element.

- getQuarter(int[][] arr, int quarter): Extract a specified quadrant from the array.

- Calculate the starting row and column based on the quadrant number.

- Copy the elements from the original array to the new quadrant array.

## Screenshots for Task C

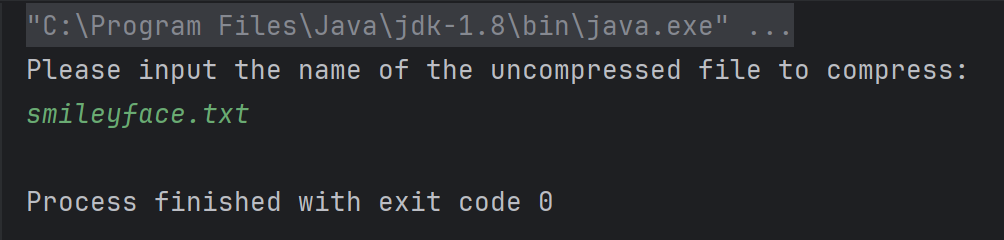


Figure 3: Terminal Output for Running ImgQuadTreeFileCreatorDriver

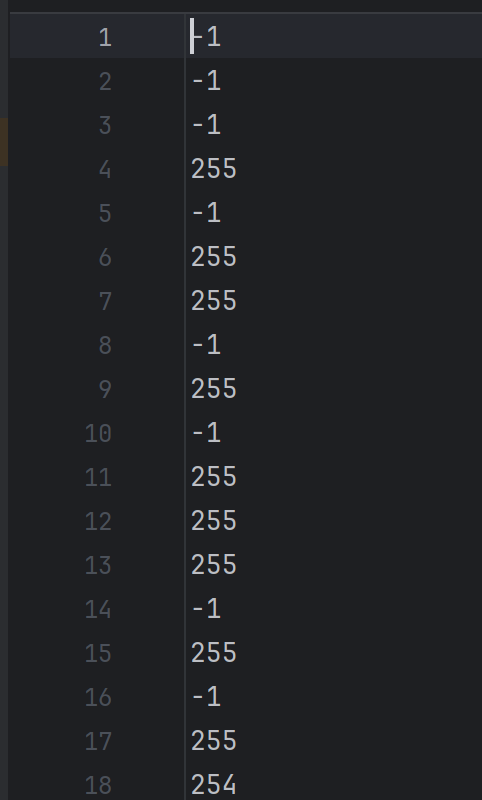


Figure 4: A View of the Resulted File

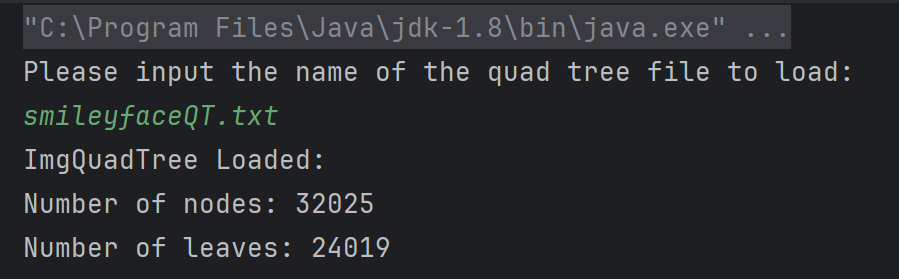


Figure 5: Terminal Output for Running the Resulting File with ImgQuadTreeDisplayer

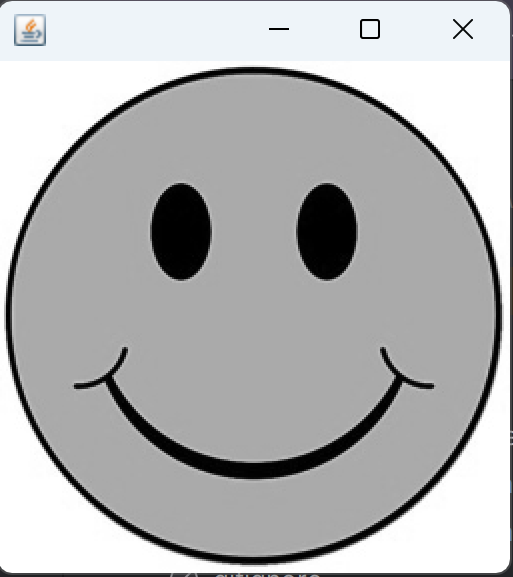


Figure 6: The Resulting Picture by Running ImgQuadTreeDisplayer with the Resulting File