ICS202-Lab Project

(It is an individual project and Submission is due on 11/05/2024)

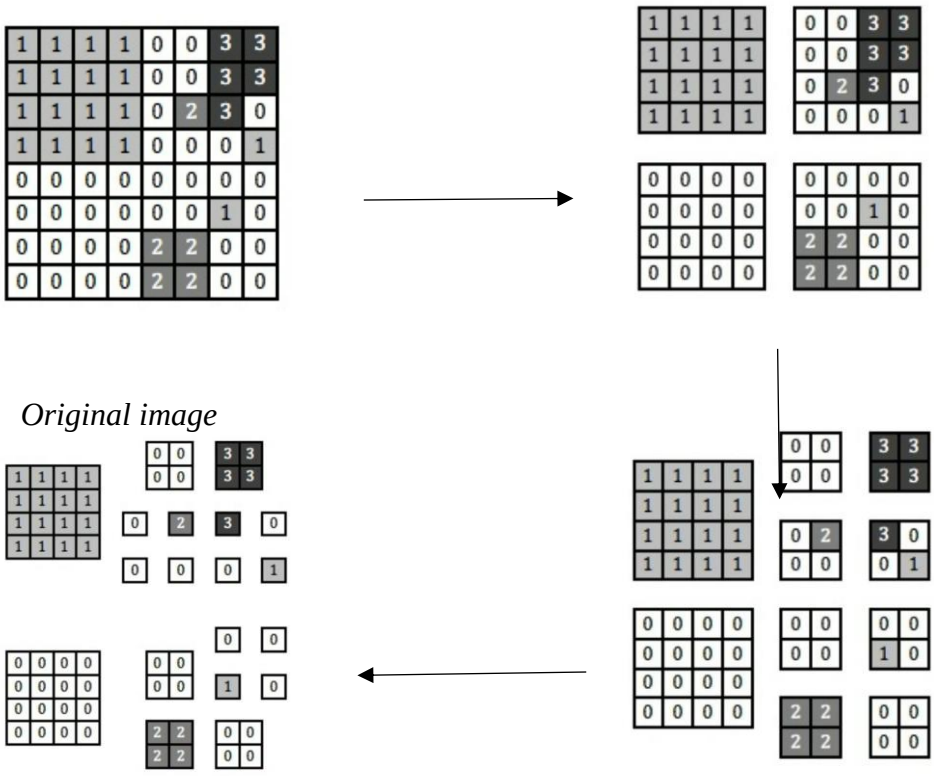
What is Quadtree?

The quadtree data structure is a sort of tree that may recursively partition a flat 2-D space into four quadrants. Each hierarchical node in this tree structure has zero or four children. It is suitable for a variety of applications, including sparse data storage, image processing, and spatial indexing. In simple terms, a quadtree is a tree where each node is either a leaf or a non-leaf with 4 children.

Quadtree in Image Processing

Computer systems use a matrix of pixel values to represent images. Red, Green, and Blue (RGB) values are the integers that correspond to the intensity of the composition of each pixel. There are many different color options available, with values ranging from 0-255.

From the perspective of image processing, the quadtree is a method of representing images that divides an array into quadrants one after the other. Essentially, the array is divided into quadrants again and again until blocks—which could be single pixels—are generated that are made up entirely of the same kind of pixels. Consider grayscale image consists of pixels with intensities ranging from 0 to 255 (inclusive). We can store a picture in a compressed format using a quadtree. Assume that the image region is square and the number of pixels across (and down) is a power of two, as illustrated below, where pixel intensities are 0,1,2,3. If the image does not consist of pixels of the same intensity, then the image is divided recursively into four quadrants (which are also squares) until each square consists of a single intensity, as shown by the sequence below:

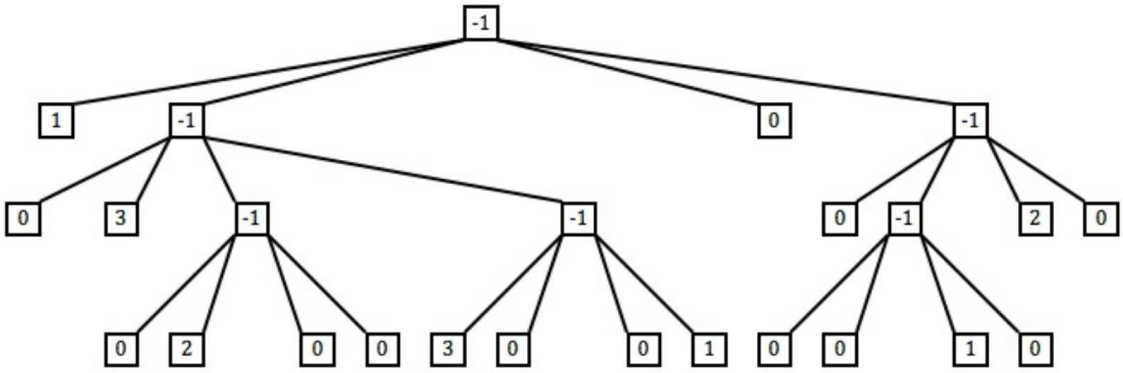


Original image

Final division

Now the image is stored as a quadtree where the root represents the whole image. If the image is split into 4, the node stores a -1 and has four children representing the image's four quadrants (in the order NW,

NE, SW, SE). If the quadrant consists of 1 color only, then the node is a leaf and stores the intensity of the quadrant. This process is repeated for each quadrant if necessary. Based on this information, the image above would be stored as a quadtree as follows:



The quadtree can then be stored in a file, one integer per line, by performing a preorder traversal of the tree:
-1 1 -1 0 3 -1 0 2 0 0 -1 3 0 0 1 0 -1 0 -1 0 0 1 0 2 0

Notice the number of required integers for the representation of the above image which is 8 by 8. When the quadtree mechanism is applied, only 25 integers whereas the original image would have required 64 integers (since it is an 8 X 8 image). This helps a lot in reducing the image size. Read on for the tasks you need to complete.

Tasks (Read carefully until you understand)

Considering a 256 X 256 image, the task is to complete an `ImgQuadTree` class (`ImgQuadTree.java` given with this project) to generate a quadtree that represents the image. The `ImgQuadTree` class requires an inner class to represent a node of the quadtree. Each node consists of an intensity value and references to four children. The intensity value can be an integer between 0 and 255 (if the node is a leaf) or -1 (if the node is a non-leaf). The inner class should have a constructor that creates a node with no children and an intensity value of -1. These can be changed once the node is inserted into the tree.

(a) Complete the constructor for the `ImgQuadTree` class that has one parameter, the name of a text file (*walkingtotheskyQT.txt*) containing the preorder traversal of the quadtree for an image, like the example shown above. The file will have one integer per line, and each integer will be between -1 and 255, inclusive. You may assume that the input file is the preorder traversal for a valid quadtree. You should read the data from the file and build a quadtree based on the data. In case, the file cannot be opened, just output an error message and exit the program.
   (i) Complete the method `getNumNodes` that returns the number of nodes in the quadtree. Do this as efficiently as possible.
   (ii) Complete the method `getNumLeaves` that returns the number of leaves in the quadtree. Do this as efficiently as possible.
   (iii) Complete the method `getImageArray` that creates an array of size 256 X 256 and then traverses the quadtree, filling in each cell of the array with its intensity value. Return this array when you're done. Note that as you traverse the quadtree, whenever you hit a leaf, you may need to fill in more than one array cell with the leaf's intensity value. (*Should use recursion here*.)

(b) Once you complete the `ImgQuadTree` class, run the `ImgQuadTreeDisplayer` (given with this project) provided for you to load in a sample quadtree file (*walkingtotheskyQT.txt*) and make sure you can see the original image. This application passes the quadtree file name to your constructor so you can create the tree, then it calls your `getImageArray` method to get the image array based on your quadtree and displays the image in a window. Keep testing until you have your `ImgQuadTree` class working correctly.

(c) Finally, write another class named `ImgQuadTreeFileCreator` that reads in the name of a text file (*smilyface.txt*) containing the data for an uncompressed image of size 256 X 256 (one integer per line), and creates a new text file containing the data needed for the quadtree representation for the image. If you write this class correctly, you can use the generated quadtree data file with the `ImgQuadTreeDisplayer` above to view the image. Note: This can be done without creating a quadtree. (*Use recursion here*). See below for a sample image file that you can convert to quadtree files.

Test File Format

The quadtree files consist of one integer per line, representing the preorder traversal of a quadtree that represents a 256 X 256 grayscale image consisting of pixel intensities between 0 and 255 (inclusive) for leaf nodes and -1 values for non-leaf nodes. You may assume that the quadtree files are valid. The *walkingtotheskyQT.txt* is a sample quadtree file you can use for testing purposes.

The image files consist of one integer per line, representing the raw (uncompressed) data for the image. The integers are in *row-major* order, meaning that the image is scanned row by row. So the first 256 integers represent the first row of the image. The next 256 integers represent the 2nd row of the image, etc. The integers are always in the range 0 to 255 (inclusive*).* The *smilyface.txt* is a sample image file you can use for testing purposes.

Submission Details:

- ✓ Submit all Java files.
- ✓ Use comments in your code
- ✓ Cheating and copying are strictly prohibited and evidence of it will result in a zero grade.
- ✓ A report containing the following:
  - Introduction
  - Your problem-solving strategy for each task and subtasks mentioned above
  - Test case screenshots for converting (i) from quadtree file to image, and (ii) from image to quadtree file
  - The challenges/difficulties you face in implementing the project
- ✓ Zip all the items in a folder and named as *YourkfupmID_Sec#*
- ✓ Consider the points mentioned in the rubrics below:

Performance Indicator	Exemplary (100)	Satisfactory (75)	Developing (50)	Unsatisfactory (25)
PI(1): Design capabilities	Design follows the requirement and considerations for space and time efficiency is evident.	Design follows the requirement but no considerations for space and time efficiency is evident.	Some parts are designed by considering the problem/requirement, but not all.	No evidence of data structure and solution design based on the problem/requirement.
PI(2): Implementation	§ Modular Code with proper encapsulation	One of the following issues:	More than one of the following issues:	§ Code not modular with no proper encapsulation
	§ Proper identity (variables) and behavior (methods)	§ Code not modular with no proper encapsulation	§ Code not modular with no proper encapsulation	§ Ill-structured code
	§ Proper use of structured constructs (if, while, for, etc.)	§ Ill-structured code	§ Ill-structured code	§ Not that well indented
	§ Well indented	§ Not that well indented	§ Not that well indented	§ Some vague variable and method names
		§ Some vague variable and method names	§ Some vague variable and method names	
PI(3): Evaluation	⍰ Compiles without warnings or errors.	⍰ Compiles without warnings or errors.	⍰ Compiles without warnings or errors.	May only run for few cases or does not run/compile.
	Runs for all cases.	Runs for the general cases,	Runs for general cases.	
		Misses at most two special cases.	At least three or more different cases do not run as per the given requirements.	