Assignment 4

CS 1027
Computer Science Fundamentals II

Due date: April 5 at 11:55 pm

Learning Outcomes

In this assignment, you will get practice with:

- Implementing trees
- Designing algorithms to traverse and manipulate information stored in trees
- Use trees in a simple application to display images at different resolutions

Introduction

An image can be viewed as a 2-dimensional matrix of pixels which can be organized in a hierarchical manner using a special kind of tree called a *quadrant tree*. In a quadrant tree the root of the tree represents the whole image. The image is divided into four regions or quadrants of the same size and each quadrant is represented by a node that is placed as a child of the root. Each quadrant Q can be recursively split into 4 quadrants, each represented by a node that is placed as a child of the node representing Q. This partitioning continues until each quadrant contains a single pixel. The following figure shows an example of the first 2 levels of a quadrant tree.

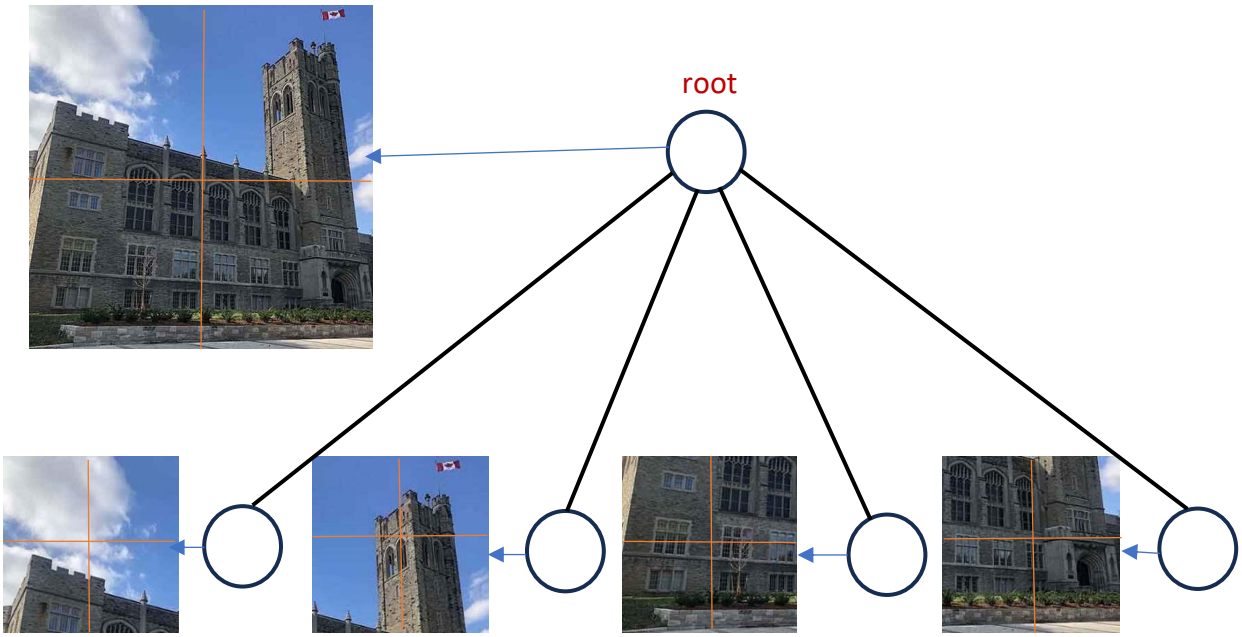


Figure 1.

Quadrant trees have many applications. For example, they can be used to compress the size of an image by storing it at different resolutions as follows. Every node of the quadrant tree represents a region of an image; if we store in each node the average color of the pixels in its region, then we can compress an image by only saving the pixels stored in the nodes at a given level of the tree. So, if the quadrant tree representing an image has

a height of h, then saving all the pixels stored in the nodes at level h of the tree would save the image at its initial resolution. Saving only the pixels in the nodes at level h-1 would decrease the size of the picture by a factor of 4 (as only one pixel will be saved for each region of size 2 × 2), but the resolution of the saved picture will also be reduced. The following images show the same figure saved at different resolutions.

Figure 2.

In this assignment you will implement a quadrant tree, as specified below, and you will use it in a simple application that allows you to change the resolution at which an image is displayed.

Provided Files

- ListNode.java is a class for nodes in a singly linked list.
- Duple.java has two instance variables:
  - front is a reference to the first node of a singly linked list
  - count is the number of nodes in the list
- TestQuadrant.java is a tester file to **help** check if your java classes are implemented correctly. Similar tester files will be incorporated into Gradescope's auto-grader. Additional tests will be run that will be hidden from you. **Passing all the tests within the provided files does not necessarily mean that your code is correct in all cases.**
- QTreeException.java.
- CheckTree.java can be used to check that you build the quadrant tree correctly. It prints the information stored in the nodes of the quadrant tree shown in the document ExampleTree.pdf.

Carefully read the code of the above java classes so you understand the methods that they provide. The following java classes are used to display the graphical user interface and your code will not need to use them: Board.java, DrawImage.java, Gui.java.

Assignment 4

CS 1027
Computer Science Fundamentals II

Classes to Implement

For this assignment, you must implement the Java specified below. In these classes, you may implement more private (helper) methods if you want. However, you may not implement more public methods. You may **not** add instance variables other than the ones specified in these instructions nor change the variable types or accessibility (i.e. making a variable public when it should be private). Penalties will be applied if you implement additional instance variables or change the variable types or modifiers from what is described here.

QTreeNode.java

This class represents a node of the quadrant tree. This class must have the following private instance variables:

- int x, y represent the coordinates of the upper left corner of the quadrant represented by this QTreeNode object
- int *size* is the size of the quadrant. All quadrants are square; so, the coordinates of the vertices of the quadrant represented by this object are (x,y), (x+size-1,y), (x,y+size-1), and (x+size-1,y+size-1).
- int *color* is the average color of the pixels stored in the quadrant represented by this object
- QTreeNode parent is the parent of this QTreeNode object
- QTreeNode[] children stores the children of this QTreeNode object. An internal node has 4 children; a leaf node has no children

In this class you must implement the following methods.

- public QTreeNode(): This is the constructor for the class. This method initializes parent to null and creates an array of size 4 for children, setting every entry to null. All other instance variables are initialized to 0.
- public QTreeNode(QTreeNode[] theChildren, int xcoord, int ycoord, int theSize, int theColor): A second constructor for the class. Initializes the instance variables to the values passed as parameters.
- public boolean contains(int xcoord, int ycoord): This method returns true if the point (xcoord,ycoord) is contained inside the quadrant represented by this object; it returns false otherwise.
- public int getx(), int gety(), int getSize(), int getColor(), QTreeNode getParent(): Getter methods for the class
- public QTreeNode getChild(int index) throws QTreeException: Returns the child stored at the specified index of array children; throws a QTreeException if children is null or if index < 0 or if index > 3,

- public void setx (int newx), sety(int newy), setSize(int newSize), setColor(int newColor): Setter methods for the class.
- public void setParent(QTreeNode newParent): Sets the parent of this object to the specified value
- public void setChild(QTreeNode newChild, int index): Sets children[index] to newChild; throws a QTreeException if children is null or if index < 0 or if index > 3.
- public boolean isLeaf(): returns true if children is null or if every entry of children is null; returns false otherwise.

QuadrantTree.java

This class must have the following private instance variable:

- QTreeNode root: The root of the quadrant tree

In this class you must implement the following methods.

- public QuadrantTree (int[][] thePixels): Builds a quadrant tree corresponding to the pixels in the given 2-dimensional array thePixels. Array thePixels has thePixels.length rows and thePixels.length columns. You can assume that thePixels.length is a power of 2.
  To build the tree you must use a recursive algorithm. Your algorithm could work as follows:
  - The root node R is a QTreeNode representing the following quadrant Q:
    - the upper-left corner of Q is at position (0,0)
    - the size of Q is thePixels.length
    - the color of Q is the average color of the pixels in Q. To get the average color of the pixels invoke method Gui.averageColor; this method receives as parameter the 2-dimensional array thePixels, the coordinates x,y of the upper-left corner of Q and the size of Q and it will return the average color of the pixels in Q.
  - The parent of R is null.
  - If the size of Q is equal to 1 then R has no children (so instance variable children in R is null)
    if the size of Q is larger than 1, then R will have 4 children:
    - The first child R1 is a QTreeNode corresponding to the upper-left sub-quadrant Q1 whose upper-left corner is at the same position as the upper-left corner of Q, its size is half of the size of Q, its color is the average color of the pixels in Q1, its parent is the node corresponding to the quadrant Q. If the size of Q1 is 1 then R1 has no children, otherwise R1 has 4 children created similarly as how the children of R are created.
    - The other children of R are created in a similar manner: The second child R2 corresponds to the upper-right sub-quadrant of Q, the third

child R3 corresponds to the bottom-left sub-quadrant of Q, and the fourth child R4 corresponds to the bottom-right sub-quadrant of Q.

The following figure shows an example of the construction of a quadrant tree for a 2-dimensional array of size 4 × 4.

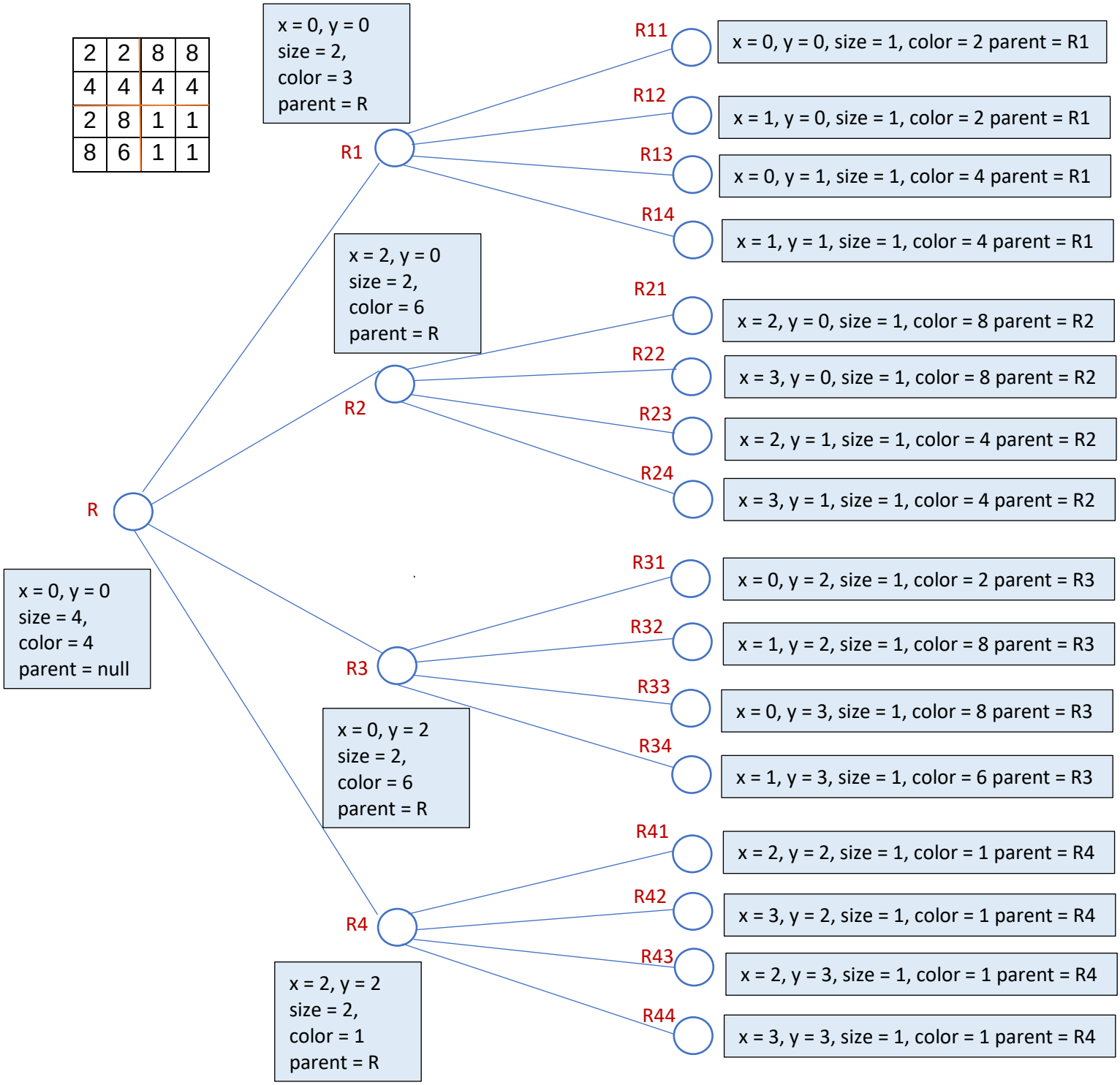


Figure 3.

- public QTreeNode getRoot(): Returns the root of this quadrant tree.
- public ListNode<QTreeNode> getPixels(QTreeNode r, int theLevel): Returns a list containing all the nodes in this quadrant tree at the level specified by theLevel; if theLevel is larger than the height of this tree, then return a list with all the nodes at the last level of the tree. This method must be recursive.
  Read the class ListNode to learn how to create a linked list storing QTreeNode objects.
  For example, for the following tree, if theLevel = 1 then the algorithm must return a list storing the nodes B, C, D, E, and if level = 3 the algorithm must return a list storing the nodes F, G, H, I, J, K, L, M, N, O, P, Q, R, S, T, U

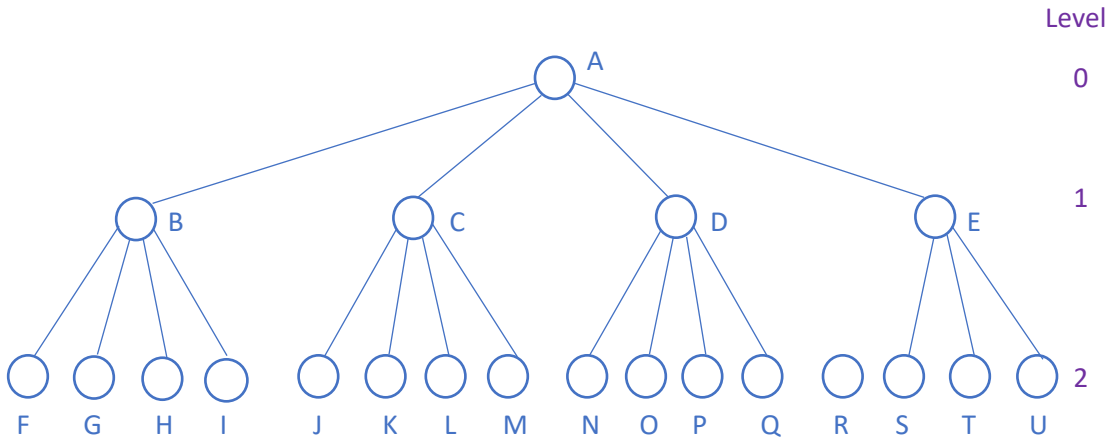


Figure 4.

Hint. Perform a traversal of the tree: If r is a leaf or theLevel = 0 create and return a list storing r. Otherwise, perform recursive calls on the children of r concatenating the lists that these calls return (what are the values of the parameters in these calls?); return the final list.

- public Duple findMatching (QTreeNode r, int theColor, int theLevel): Returns an object of the class Duple storing two values: (1) a list of nodes of the class ListNode<QTreeNode> containing all the nodes in this quadrant tree that have color similar to theColor and either are at the level specified by theLevel or appear at the last level of the tree if theLevel is larger than the height of the tree; and (2) the number of nodes in this list. To determine whether the node referenced by r has a color similar to theColor, you can invoke method Gui.similarColor (r.getColor(), theColor). This method must be recursive.
  Hint. Perform a traversal of the tree: If r is a leaf or theLevel = 0 then (i) if the color of r is similar to theColor return a Duple object storing a list containing r and the

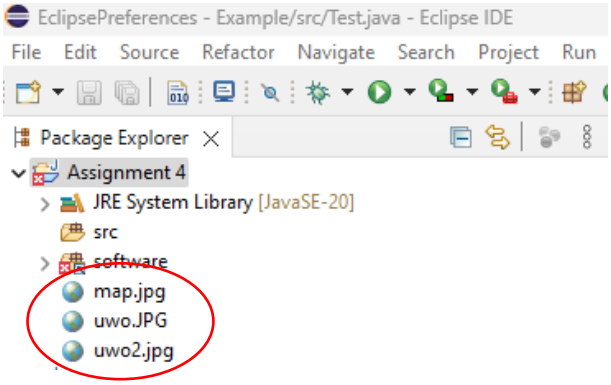
value 1; (ii) what value do you return if the color of r is not similar to theColor? If r is not a leaf or theLevel > 0 then perform recursive calls on the children of r concatenating the lists inside the Duple objects that these calls return (what are the values of the parameters in these calls?); return the final Duple object (what is the value of the instance variable *count* of this Duple object?).

- public QTreeNode findNode(QTreeNode r, int theLevel, int x, int y): Returns a node in the subtree rooted at r and at level theLevel representing a quadrant containing the point (x,y); it returns null if such a node does not exist. This method must be recursive.
  For example, for the tree in Figure 3 with r = R, level = 1, x = 3, and y = 1, the method must return the node R2. In the same figure, if r = R, level = 2, x = 4, and y = 1, the method must return null.

Running the Program

You are provided with a simple graphical user interface that uses your code to display an image, change its resolution, and find pixels with similar colors. To run the program from the terminal, place all the java files and images in the same directory. Compile the program by typing: javac Gui.java. Then run the program by typing: java Gui file_name, where file_name is the name of an image file. The image file must have size at least 512 × 512 and at most 2048 × 2048.

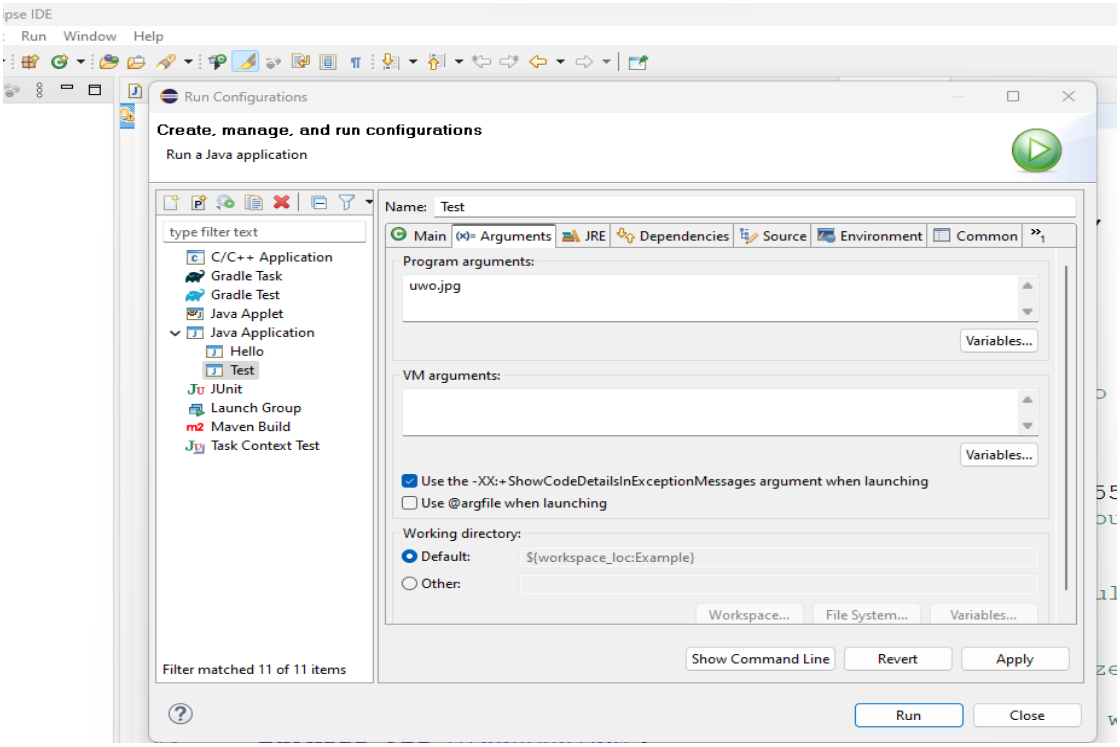
If you wish to run the program from Eclipse, you need to place the image files in the root directory of your project, not inside the src folder:



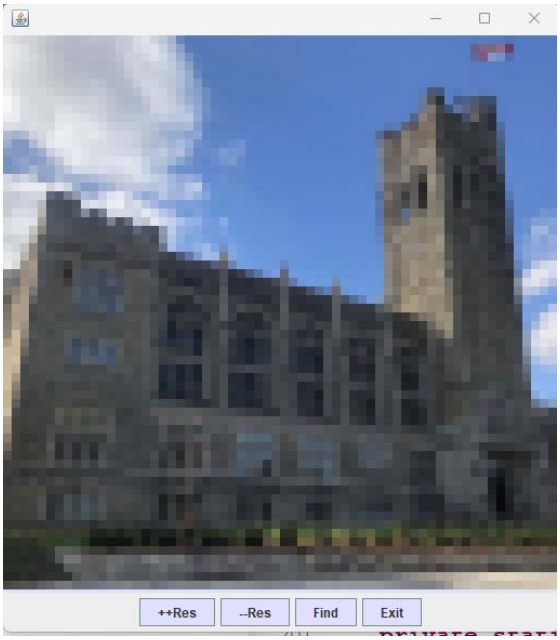
Then in Eclipse you must select Run → Run Configurations, choose the tab Arguments and in Program Arguments enter the name of the image file you wish to use:

Assignment 4

CS 1027
Computer Science Fundamentals II



The following figure shows a screenshot of the program:



Clicking on ++Res increases the resolution of the image, --Res decreases the resolution; if the image is of size larger than 512 × 512, once the displayed image is at full resolution, clicking on any pixel of the image and then selecting ++Res will zoom in on the quadrant

containing the selected pixel. Clicking on a pixel and then selecting Find will show all pixels with a similar color as the selected pixel.

Marking Notes

Functional Specifications

- Does the program behave according to specifications?
- Does it produce the correct output and pass all tests?
- Are the classes implemented properly?
- Does the code run properly on Gradescope (even if it runs on Eclipse, it is **up to you** to ensure it works on Gradescope to get the test marks)
- Does the program produce compilation or run-time errors on Gradescope?
- Does the program fail to follow the instructions (i.e. changing variable types, etc.)

Non-Functional Specifications

- Are there comments throughout the code (Javadocs or other comments)?
- Are the variables and methods given appropriate, meaningful names?
- Is the code clean and readable with proper indenting and white-space?
- Is the code consistent regarding formatting and naming conventions?
- Submission errors (i.e. missing files, too many files, etc.) will receive a penalty.
- Including a "package" line at the top of a file will receive a penalty.

Remember **you must do** all the work on your own. **You cannot copy code, share code, use code from online or other sources, use a chatbot, pay someone to write your code**. All submitted code must be 100% yours. All code will be run through similarity-detection software.

Submission (due April 5 at 11:55 pm)

Assignments must be submitted to Gradescope, not on OWL. If you are new to this platform, see these instructions on submitting on Gradescope.

Rules

- Please only submit the files specified below.
- Do not attach other files even if they were part of the assignment.
- Do not upload the .class files! Penalties will be applied for this.
- Submit the assignment on time. Late submissions will receive a penalty of 10% per day.
- Forgetting to submit is not a valid excuse for submitting late.
- Submissions must be done through Gradescope. **If your code runs on Eclipse but not on Gradescope, you will NOT get the marks!**
- You are expected to perform additional testing (create your own tester class to do this) to ensure that your code works for a variety of cases. We are providing you with some tests but we may use additional tests that you haven't seen for marking.
- Assignment files will not be marked if emailed to the instructor(s) or TA(s).

- You may re-submit code as many times as you wish, however, re-submissions after the assignment deadline will receive a late penalty.

Files to Submit

- QTreeNode.java
- QuadrantTree.java

Grading Criteria

Total Marks: [20]

Functional Specifications:

[1 mark] QTreeNode.java

[5 marks] QuadrantTree.java

[12 marks] Passing Tests (some additional, hidden tests will be run on Gradescope)

Non-Functional Specifications:

[0.5 marks] Meaningful variable names, private instance variables

[0.5 marks] Code readability and indentation

[1 mark] Code comments