CPSC 449 Assignment 2

Due: Tuesday June 19, 2018 at 11:59pm
Sample Solution Length: About 90 lines (in addition to the provided code) without any comments

Individual Work:

All assignments in this course are to be completed individually. Students are advised to read the guidelines for avoiding plagiarism located on the course website. Students are also advised that electronic tools may be used to detect plagiarism.

Late Penalty:

Late assignments will not be accepted.

Background:

A quad tree is a data structure that can be used to store data with a two-dimensional structure to it, such as an image. As the name suggests, a quad tree has internal nodes that have exactly four children. Each leaf node contains data for the region that it represents, such as the color of the region within an image. When an image contains large homogenous regions a quad tree can be a spatially efficient mechanism for representing an image (however, when the image doesn't contain such regions a quad tree isn't an efficient representation). While quad trees can be used to represent rectangular regions, we will confine ourselves to square images where the width and height of the image are powers of 2 so that the number of special cases that need to be considered is kept to a minimum.

Provided Code:

Some starter code has been provided on the course website in quadtree_start.hs. The comments in that file indicate where you should put your code for the various parts of the assignments. That file also contains a main program that calls the functions you will write, but the function calls themselves have been commented out with comments of the form `-- Part n --`. These comments should be removed as you begin work on each part of the assignment so that the relevant lines in the main program actually call the function that you are writing. Don't modify the provide code other than to add your name and student number to the top of the file, insert your code at the indicated locations, change the files that are being loaded, change the merging function being used, or to remove the comments described previously.

Part 1:

Define a recursive algebraic type for a quad tree. The base case will be a leaf node that contains a Color. The Color type is already defined in the provided code on the course website. It is a product type that contains 3 integers where the first Int represents an amount of red, the second represents an amount of green and the third represents an amount of blue. Each color level will be an integer value between 0 and 255. The recursive case is an internal node that has four children. No data is stored in the internal nodes.

While a quad tree describes the structure of an image, it doesn't describe its size. As a result, you will need to define a second algebraic type for an image that consists of an integer and a quad tree. The integer will represent the width (and height) of the image (since it is square) and the quad tree will contain the color data for the image which will then be scaled by the width value when it is displayed.

Part 2:

I have provided code that loads a PNG image file and stores it in a two dimensional list. The two dimensional list has one entry in it for each pixel in the image. Each pixel is of type Color (and as such stores the red, green and blue components of the pixel). The type of such an image is [[Color]]. The pixels in the image are stored in rows from the top of the image to the bottom of the image, and the pixels in each row are stored from left to right.

Write a function named `createTree` that takes a list representation of an image and stores it in a quad tree image. Your function will take one parameter of type [[Color]], and it will return one result that is a quad tree image (the second type that you defined in Part 1 consisting of both the image's width and the tree). Your function should report an error (and quit your program) if the provided image is not square or if the width or height is not a power of 2.

A general approach that works for solving this problem is to examine the current image. If it's homogenous, meaning that all of the pixels are the same color, then create a leaf node of that color. Otherwise, break the image into 4 smaller square sub-images (all of which will have the same size) and construct an internal node in the quad tree whose children are populated by recursively constructing four child nodes for the four smaller sub-images.

The tree that is returned by createTree should be an efficient representation of the image, meaning that it should not use more nodes that necessary. Specifically, it is **not** acceptable to construct a tree where each pixel in the image is represented by a different leaf node.

Part 3:

Write a function named `countNodes` that determines how many internal nodes and how many leaf nodes there are in a quad tree image. Your function will take a quad tree image as its only parameter and return a tuple containing two integers. The first integer in the tuple will be the number of internal nodes in the tree. The second integer in the tuple will be the number of leaf nodes in the tree.

Your countNodes function will help you verify that you are generating the quad tree images in Part 2 correctly. Counts of internal and leaf nodes for each of the provided images can be found on the course website. I'd recommend that you **not** move on to the later parts of the assignment until your counts match what is on the website.

Part 4:

Write a function named `toHTML` that generates the HTML SVG tags necessary to render a quad tree representation of an image in a browser. To do this, traverse your tree so that you visit every leaf node and generate a <rect> tag for each leaf node that fills the correct region with the correct color. The order in which the tags are generated is not important because none of the rectangles that you are generating should overlap.

Part 5:

Write a function named `merge` which merges two quad tree images. Your merge function will take three parameters and return a quad tree image as its only result. The first two parameters to merge will be the quad tree images that are being merged. The third parameter will be a function that merges two colors, returning a color as its result (which means that merge is a higher order function). The result of the merge function should be a new image where overlapping pixels have been merged using the merge

function provided as the third parameter. Your merge function should report an error (and quit your program) if the images that are being merged have different sizes.

The following general approach can be taken to implement the merge function:

- If the current node in tree #1 is an internal node and the current node in tree #2 is an internal node then construct a new internal node for the result by recursively merging the corresponding child nodes in each tree.
- If the current node in tree #1 is a leaf node and the current node in tree #2 is a leaf node then construct a new leaf node for the result by merging the colors of the two leaf nodes using the color merging function provided as the third parameter.
- If the current node in one tree is an internal node and the current node in the other tree is a leaf node then construct a new internal node for the result by recursively merging the leaf node with each of the four children of the internal node.

Three merge functions are included in the provided code: average, brightest and darkest. You can write additional merge functions if you want to (for example, a slightly more complex merge function could be used to perform green screening) but there is no requirement to do so.

Part 6:

When two images are merged the resulting quad tree may not be optimal. In particular, there could be interior nodes in the tree where all four children are leaf nodes that contain the same color. For example, such a situation arises when blocks_1.png and blocks_2.png are merged. The resulting quad tree image contains one internal node and 4 leaf nodes, all of which are the same color. Most of these nodes are unnecessary because the same image can be represented by a single leaf node.

Create a function named `optimizeImage` that optimizes a quad tree image by detecting cases where an internal node has four children, all of which are leaf nodes of the same color, and reducing that collection of nodes to a single leaf node. Your optimization function should traverse the tree in post-order so that any new redundancies introduced during the merge process are also removed (such as what occurs when blocks_3.png and blocks_4.png are merged). Your function should not make any changes to the image unless it contains one or more nodes that have 4 leaf nodes that are the same color.

For an Additional Challenge:

Extend your implementation of all of the functions so that they correctly handles rectangular images of arbitrary width and height (you can no longer assume that the width and height are the same, or that they are powers of 2). You'll need to change the algebraic type defined in Part 1 so that the type that stores the image includes both the width and height, along with the implementations of your functions for working with the quad tree representation of an image to complete this part of the assignment.

Grading:

A base grade will be determined for your assignment based on its overall level of functionality, as shown below:

A+: All parts of the assignments are completed successfully, included the additional challenge.

A: All parts of the assignment, except for the additional challenge, are completed successfully.

B+: The data types are defined correctly, and four of the five functional components work correctly (converting a list representation of the image to a tree, counting the nodes, merging two trees, optimizing a tree, and converting a tree to a collection of tags), along with a credible attempt at the final functional component.

B-: The data types are defined correctly, and three of the five functional components work correctly (converting a list representation of the image to a tree, counting the nodes, merging two trees, optimizing a tree, and converting a tree to a collection of tags), along with a credible attempt at the functional components that do not work correctly.

C: The data types are defined correctly, and two of the five functional components work correctly (converting a list representation of the image to a tree, counting the nodes, merging two trees, optimizing a tree, and converting a tree to a collection of tags), along with a credible attempt at the functional components that do not work correctly.

D / F: Submissions that do not meet the standard for a C will be awarded a grade of D+, D or F depending on functionality, quality and quantity of the submitted code.

Once your base grade has been established it may be reduced if your implementation fails to meet the specifications outlined in this document, fails to use functional programming constructs in a reasonable manner, has stylistic deficiencies, or otherwise behaves in an undesirable manner. Examples of stylistic deficiencies and other undesirable behaviour include (but are not limited to):

- Repeated code
- Magic numbers
- Missing or low quality comments
- Poor function / parameter names
- Crashing
- Generating useless output (such as 0 width or 0 height rectangles)
- Unnecessarily complex solutions