CPSC 449 Assignment 2

Due: Friday November 17, 2017 at 12:00 noon
Sample Solution Length: About 80 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.

Part 1:

Define a recursive algebraic type for a quad tree. The base case will be a leaf node that contains an (Int, Int, Int) tuple where the first Int represents an amount of red, the second represents and amount of green and the third represents an amount of blue. Each color level will be a value between 0 and 255. The recursive case is an internal node that has four children. No data is stored in the internal nodes.

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.

Part 2:

I have provided code that loads a PNG image file and stores it in a two dimension lists. The two dimensional list has one entry in it for each pixel in the image. Each pixel is represented by an (Int, Int, Int) tuple consisting of the red, green and blue components of the pixel. The type of such an image is [[(Int, Int, Int)]].

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 [[(Int, Int, Int)]], 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.

Hint: 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 by recursively constructing four child nodes for the four smaller sub-images.

Part 3:

Write a function named treeMap that maps a function over all of the nodes in a quad tree, returning the modified quad tree as its result. The treeMap function will take a function, f, as its first parameter, and it will take a quad tree, q, as its second parameter. The function f will take a quad tree as its only parameter, returning a quad tree as its result.

Create a function named grayscale that converts an image to gray scale (by averaging the red, green and blue values in each leaf node and then using that average value for all three color components). Create a second function named mirror that mirrors the image (left and right are swapped). Each of these transformations must be implemented by mapping a function over the quad tree using your treeMap function. A image stored in a quad tree can be mirrored by changing the order of the children for each non-leaf node while leaving the leaf nodes unchanged. A grayscale image can be created by modifying the values in leaf nodes while leaving the non-leaf nodes unchanged.

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.

For an Additional Challenge:

Extend your implementation so that it 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 the width and height, along with the implementations of your functions for working with a quad tree representation of an image when completing 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 two of the three major functional components work correctly (converting a list representation of the image to a tree, the grayscale and mirror functions, and converting a tree to a collection of tags), along with a credible attempt at the third major functional component.

C: The data types are defined correctly, and one of the three major functional components works correctly, along with a credible attempt at the other two components.

D / F: None of the major functional components consistently provides correct results.

Once your base grade has been established it may be reduced if your implementation fails to use functional programming constructs in a reasonable manner, has stylistic deficiencies or other undesirable behaviour. 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)