Drawing trees

Help Center

In the supplied `Tree` class, we included example code that draws a rooted tree in its standard configuration with the root node at the top. In this practice activity, we review this drawing code and consider how drawing a tree can be expressed as a recursive process.

Computing the bounding box for a tree

Prior to drawing a tree, having an estimate of the canvas space required to draw a tree is very helpful. For this purpose, our first task is compute the height and width of a rectangular box that bounds the diagram of the tree. This box is referred to as the *bounding box* for the tree. We can then use the size of this box to determine the size of the required canvas. Note the size of this box can be computed directly from the height of the tree and the number of its leaves. However, we will compute these values recursively from the structure of the tree to provide more practice with recursion and trees.

In the base case, a tree consists of a single node. The constants `NODE_HEIGHT` and `NODE_WIDTH` represent the user-defined height and width of the bounding box for a single node. At this point, you are welcome to implement a recursive function `get_box_size(tree)` that takes a tree and returns the height and width of the bounding box of this tree using these constants. Our implementation of `get_box_size` is a method in the `TreeDisplay` class in `poc_draw_tree`. Feel free to use our implementation as a template for your efforts. Just hide the body of `get_box_size` using code folding while you reimplement it.

In the case where the root node has one or more subtrees, our solution computes the height and width of the bounding boxes for each subtree recursively. In this case, the height of the bounding box for the entire tree is `NODE_HEIGHT` plus the maximum of the heights of the bounding boxes for each subtree. The width of the bounding box for the entire tree is the larger of the width of the bounding box for the root and the sum of the widths of the subtrees.

Drawing a tree

When drawn as a diagram, a tree consists of a collection of nodes and edges. For rooted trees, this diagram can be drawn using a recursive method `draw_tree` with two parts:

- Draw the root node and the edges from this root node to the roots of its children,
- Call `draw_tree` to draw the diagrams for each of the root node's subtrees.

One subtle issue with `draw_tree` is that the line segments from the root node to its children should be drawn before the circles for these nodes. This ordering ensures that the edges aren't drawn on top of the nodes in the tree. This observation leads to the conclusion that these edges must be drawn before any recursive calls to `draw_tree` happen. However, the `draw_line` statements for these edges still require the positions of the root nodes's children.

Our solution to this problem is to call `get_box_size` on each of the subtrees prior to any drawing. With this information, we can compute the centers for all of the children of the root node and use `draw_line` to draw the required edges. After one call to `draw_circle` to draw the root node, we can then recursively call `draw_tree` on each subtree. For the calls to `draw_circle`, we used a diameter that is half of `NODE_HEIGHT`.

To finish this activity, you are welcome to attempt an implementation of `draw_tree(canvas, tree,`

`pos)` using your knowledge of SimpleGUI. The parameter `pos` should correspond to the location of the upper left corner of the tree's bounding box on the canvas. Again, we recommend that you use our code as a template and simply hide the body of `draw_tree` using code folding while you are reimplementing it.

Created Wed4 Jun 2014 11:26 PM BST

Last Modified Fri 10 Apr 2015 7:05 PM BST