CAS CS 112 – Spring 2012 – Programming Assignment 4

Due at 10:00 pm on Thursday, March 29

1 Visualizing Binary Trees [79 pts]

In this assignment, we will augment the binary tree classes presented in the textbook with routines to create large binary trees by reading input from the user, and routines to print out binary trees in human-readable format. We will write routines to print the tree using two methods, one a text-based method described below; and the second, a graphical method that draws the trees using lines and circles on a canvas (as we will discuss in class and lab).

The assignment consists of the following components:

1. Start from the Binary Search Tree (BST) implementation in Chapter 3.

2. Augment this class to support bulk insertion of integer values. Insertions will be specified in the form of a text string consisting of a list of comma-separated integer values. For example, the string "12, 15, 76, 23, 99" should result in an insertion of 12, followed by an insertion of 15, etc. Implement this as a BST member function `void BulkInsert(String s)`. Insertions should preserve the binary search tree invariant defined in our text, and which we will discuss in more detail in lab and lecture after spring break.

3. Next, augment this class to support textual printing of the values in the trees. In order to accomplish this, we will logically assign an (x, y) coordinate to each tree node. It is appropriate for the x-coordinate of the node to be proportional to the inorder traversal number of the node in the tree, and for the y-coordinate of the node to be proportional to the the depth of the node in the tree. Therefore, we will need member variables to store these values, and routines to compute the inorder traversal number and depth of each node in the tree (you may choose what to name these). Then write a member routine `void PrintAsText()` that prints a binary tree to screen. For example, printing the binary search tree resulting from the bulk insertion above should look something like:

```
12
  15
     76
   23  99
```

4. Now, write a member routine `void PrintOnCanvas()` that prints a tree on a graphical canvas by making calls to graph-assembling functions `circle(x, y, r)`, `line(x1,y1,x2,y2)` and `text(x, y, s)` as described in lab. These functions are from the Draw package,

which is downloadable from the booksite. We will demonstrate how your program should integrate with this package in lab on Monday, March 26th. (this aspect constitutes only a small part of the overall assignment; do not wait until this date to start the rest of the assignment!).

5. Finally, write a client that tests the routines you have written. Specifically, your code should repeatedly query the user for a string corresponding to a sequence of insertions, create the binary search tree resulting from that sequence of insertions, and print this tree as text and on canvas. Prompt the user to refresh the screen and the canvasses before processing the next string.

Submit your code in two files called BST.java and BSTclient.java.

Extra credit [15 pts]: Right after BSTs, we will be covering balanced binary search trees. The primary structure we will consider is Red-Black trees. A bulk insertion into a Red-Black tree will produce a different resulting search tree than a BST. Modify your code so that it augments the Red-Black BST code provided at the booksite to print both a standard BST tree and a Red-Black tree as text and on canvas for each bulk insertion. Note that the routines you are asked to furnish are very similar for both binary search trees and Red-Black trees, so not much additional code is needed for this part (although you will have to read ahead a bit in the book).

2 Heap short answer questions [21 pts]

Please submit answers to the following questions in a text file.

1. Exercise 2.4.1 on p. 329.

2. Specify the max-heap that results when the sequence `P R I O R I T Y` is inserted an initially empty max-heap, by writing down the contents of the array.

3. Specify the max-heap that results when the sequence of insertions and deletions `P R I O R I T Y Q U E U E * * * *` is performed on an initially empty max-heap. If you need to break a tie between children, swap with the left child.