Project #5 - Writing Iterators in Ruby

Learning objectives

- Design and implement a complete program in an interpreted, object-oriented language.
- Employ sound principles of object-oriented program design.
- Write and use iterators in Ruby.
- Design and implement an efficient Ruby data structure.

Overview

The objective of this assignment is to write and use a few iterators for a new Ruby class. You will implement a binary search tree (BST) class, and write a few basic methods to manipulate objects of this class.

Specifications

Your class should be called BST. It should implement a traditional binary search tree, and should respond to the following methods:

- **`BST.new{compare_method}`** [6 pts] – return a new, empty BST. If a block of code is provided, this code is used to compare two objects in the tree during insertion, search, etc. This block should accept two parameters and return 1, -1, or 0, like Ruby's <=> operator. If no block is provided, the <=> operator is used for comparison.

- **`add(item)`** [5 pts] – add a new item to the binary tree. Add must maintain a valid binary search tree structure as new data is added to the tree. Duplicate items should be stored in the right subtree.

- **`empty?`** [3 pts] – returns true if the tree is empty. Otherwise returns false.

- **`include?(item)`** [5 pts] – returns true if the item is found in the tree, otherwise returns false. When the tree is balanced, search should take no more than O($\log_2 n$) time.

- **`size`** [4 pts] – returns the number of items in the tree.

- **`each_inorder{block}`** [6 pts] – performs an in-order traversal of the tree, passing each item found to block.

- **`collect_inorder{block}`** [6 pts] – performs an in-order traversal of the tree, passing each item found to block. The values returned by block are collected into a new BST, which is returned by collect_inorder.
- **`to_a`** [5 pts] – returns a sorted array of all the elements in the BST.
- **`dup`** [5 pts] – returns a new binary search tree with the same contents as the original tree. This operation should perform a **deep copy** of the original tree.

Grading

As in the previous lab, I will test your code with an automated test program. Your code should define your classes only. There should be no main program, and no test/debugging code in your submitted program. Running your submitted code should produce no output. I will run your code to define your classes, and then create objects and test the methods using my own code.

Your program will be graded in three areas:

1. **Capabilities and Correctness (45 pts)** – your program should correctly implement all of the above methods. It should not crash for any legal input.
2. **Readability and Style (5 pts)** – your program should demonstrate correct and efficient use of Ruby classes, logically organized, **well-commented**, and should conform to the “Coding Standards” document available on the course web page under “Handouts”.

What to Turn In

Submit a single, **well-commented** file with your Ruby code via the Pilot dropbox. The filename should be **Project5.rb**. Include **your name in the header** comments.