CS 3358 (Data Structures & Algorithms, §001&2, Fall 2024) by Lee S. Koh

Assignment 6 Part 2 (60 points)

(posted 11/13/2024, DUE as assigned on Canvas)

The Assignment

Complete the program (involving *binary search tree* and has some parts intentionally left out) contained in the supplied files.

Goal

- Hands-on programming experience on:
  - *tree* (*binary search tree* in particular)
  - *recursion*.

Relevant Suggested-book Material

Chapter 10 (Section 10.5).

IMPORTANT *Specific Requirements*

- The definition for *binary search tree* should be the one used in class (which is *different* from that adopted by the suggested-book authors).
  - ***Class definition:***
    A BST is a binary tree that (if not empty) also follows two storage rules regarding its nodes' items:
    ♯ For any node ***n*** of the tree, every item in ***n***'s left subtree (LST), if not empty, is less than the item in ***n***
    ♯ For any node ***n*** of the tree, every item in ***n***'s right subtree (RST), if not empty, is greater than the item in ***n***

    Suggested-book definition:
    A BST is a binary tree that (if not empty) also follows two storage rules regarding its nodes' items:
    ♯ For any node ***n*** of the tree, every item in ***n***'s left subtree (LST), if not empty, is less than **or equal** the item in ***n***
    ♯ For any node ***n*** of the tree, every item in ***n***'s right subtree (RST), if not empty, is greater than the item in ***n***
- `bst_insert` must be *iterative* (NOT *recursive*).
- `bst_remove` and `bst_remove_max` must use the algorithm described by the suggested book authors, appropriately adapted for the difference in tree definition above (and summarized in Slides 14 through 20 of Lecture Note ***320BinarySearchTrees***).

Your Tasks

- In `btNode.h`: provide prototypes for `bst_insert`, `bst_remove` and `bst_remove_max`.
- In `btNode.cpp`: provide definition (implementation) for `bst_insert`, `bst_remove` and `bst_remove_max`.
- Test/debug your implementation with the help of the supplied `Makefile`:
  - Do `make go` (after successful compilation or re-compilation) to test with result output to terminal.
    - Program will (display some error messages and) abort on the first detection of an incorrect outcome (associated with a certain test case).
  - Do `make gogo` (after successful compilation or re-compilation) to test with result (excluding progress-logging messages) output to file (`a6p2.out`).
    - The sample output contained in the supplied files was generated by compiling and running the program on the CS Linux server. Expect the test cases to be different if another system is used since the algorithm used for the pseudo-random number generator vary from system to system.

Deliverables

- `btNode.cpp` source file:
  - An as-is "text" version evaluator can use to compile and test.
  - A (print-to-)PDF version (be sure it jibes with the as-is "text" version) evaluator can mark on and return.

  NOTE: You don't have to include `btNode.h` as part of the ***Deliverables*** eventhough you have to fill in the prototypes for `bst_insert`, `bst_remove` and `bst_remove_max` there.

  (For testing during grading, the prototypse for `bst_insert`, `bst_remove` and `bst_remove_max` will be filled in using the *function header*s of student's implementations of the functions in `btNode.cpp`.)
  - Upload it to your ***TxSt Canvas*** account *by 11:55 pm on the due date*.

NOTES:

I Refer to this penalty tally sheet to help you minimize losing points.

(Note that it is one used in some past semester; thus some items may not be pertinent to the requirements of this semester.)

- If you discover anything that appears incorrect, please let me know (through email) as soon as possible.
- Be sure to check the homepage often for any further information related to this assignment.
