Program 6 - Binary Search Trees

CS 580U - Fall 2017
Due Date: 5:00 p.m., December 11, 2017

*Any changes/updates to the project description will be in red - **Last Updated 11/28 8:17 a.m.***
All programs will be tested on the machines in the Q22 lab. If your code does not run on the system in this lab, it is considered non-functioning EVEN IF IT RUNS ON YOUR PERSONAL COMPUTER. You can write your code anywhere, but always check that your code runs on the lab machines before submitting.

Driver Code and Test Input Files

- **Provided Files**
  - [program6.c](#) //Driver Code

Grading Rubric

TOTAL: 30 points
- **Part 1,2,3:**
  - Test 1 - Initialize the BST (1 point)
  - Test 2 - Insert into the BST (1 point)
  - Test 3 - Insert Duplicates(1 point)
  - Test 4 - Sorted data (3 points)
  - Test 5 - Search data (2 points)
  - Test 6 - Search for Missing data (2 points)
  - Test 7 - Clone and Compare trees (3 points)
  - Test 8 - Remove a value not found (1 points)
  - Test 9 - Remove Leaf (2 points)
  - Test 10 - Remove 1 child node (2 points)
  - Test 11 - Remove 2 child node with leaf (2 points)
  - Test 12 - Remove 2 child node with short circuit (2 points)
  - Test 13 - Remove 2 child root (2 points)
  - Test 14 - Remove 1 child root (2 points)
  - Test 15 - Remove Leaf root (1 points)
  - Test 16 - Clean up memory. Delete all trees (3 points)
- **Style Guidelines and Memory Leaks**
  - *You will lose significant points for the following:*
    - Makefile does not have requested format and labels (-10 points)
    - Does not pass Valgrind Tests (-5 points)
    - Does not follow requested program structure and submission format (-10 points)

Guidelines

This is an individual assignment. You must do the vast majority of the work on your own. It is permissible to consult with classmates to ask general questions about the assignment, to help discover and fix specific bugs, and to talk about high level approaches in general terms. It is not permissible to give or receive answers or solution details from fellow students.

You may research online for additional resources; however, you may not use code that was written specifically *to* solve the problem you have been given, and you may not have anyone else help you write the code or solve the problem. You may use code snippets found online, providing that they are appropriately and clearly cited, within your submitted code.

By submitting this assignment, you agree that you have followed the above guidelines regarding collaboration and research.

Part 1: BST

- Create a link based Binary Search tree composed of a Node and Tree struct. You should have a header file, bst.h, with the following:
  - Node struct containing left, right, and parent pointers, in addition to holding an Data struct value.
  - Tree struct containing a pointer to the root of the tree.
  - A function declaration for a function that allocates a tree, and initializes the root to NULL;
  - A function declaration for a function that takes a Data struct as a parameter, allocates a node, and initializes the left, right, parent fields to NULL.
- You should also have a source file, bst.c, that implements the two declared functions:
  - Tree * newTree();
  - Node * newNode(Data d, Node * parent);
- Test your functions and structure to ensure everything is initialized correctly by creating a Tree and adding a root to it.

Part 2: BST Operations

- Alter your tree struct declaration in your header file to contain the function pointers for insert, search, removeData. Implement the operations in your BST.c file.
- INSERT:
  - Create a function, Data * (*insert)(Tree * bst, Data value), that inserts into the tree – Helpful hints:
    - Return a pointer to the Data value inserted into the tree
    - Make sure you check for the special case of an empty tree [if(bst->root == NULL)],
    - After checking for the root, use a separate helper function to insert a value into the tree, Data * insertNode(Node * node, Data value), that you can use for the recursive call
    - If the value is already in the tree, return NULL
- SEARCH:
  - Create a function, Data * (*search)(Tree * bst, Data value), that searches for a value in the tree. Return a pointer to the Data object if found – Helpful hints:
    - Make sure you check for the special case of an empty tree [if(bst->root == NULL)],
    - After checking for the root, use a separate helper function to search the tree, Node * searchNode(Node * node, Data value), that you can use for the recursive call
- REMOVE:

- Create a function, void (*removeData)(Tree * bst, Data value), that removes a value from the tree – Helpful hints:
  - Use your (hopefully) working search auxiliary function to find the node you need to delete
    - Your auxiliary search function can return a node pointer, and you primary search function returns the data from that pointer.
  - You will have 3 cases requiring 3 separate functions:
    - remove a leaf node : void removeLeaf(Tree * bst, Node * d_node);
    - remove a node with 1 branch: void shortCircuit(Tree * bst, Node * d_node)
    - remove a node with 2 branches: void promotion(Tree * bst, Node * d_node)
- You will need to use your removeLeaf() and shortCircuit() functions in your promotion function, so make sure they are working before starting on the promotion function.

Part 3: Testing Your Tree

- In the driver code provided below, we do the following to test your tree:
  - Using your insert function, insert the following values into your tree (in this order)
    - `5,3,10,4,8,2,1,7,9,6,12,11,13`
- The following will be tested:
  - Insertion of the above value set
  - Search on each value
  - Search on a value not in the tree
  - Deletion of each value using the following algorithms
    - Delete leaf
    - Delete 1 child
    - Delete 2 child
    - Delete root
    - Delete 1 child root
    - Delete leaf root
- You will also need to implement the following:
  - Tree * (*clone)(Tree*): Takes a tree and uses preorder traversal algorithm to return a clone of the tree
  - int (*compare)(Tree*, Tree*): Takes a tree and uses preorder traversal algorithm to determine if the trees are equal
  - void (*sort)(Tree *, Data *): Takes a tree and a data array buffer as parameters, and fills the buffer with the tree data in sorted order using the inorder traversal algorithm.
  - void (*delete)(Tree * bst): Add a post-order deleteTree() function that deletes all nodes and the tree
    - Remember, post order only deletes leafs, so you need only call deleteLeaf()

- ○ *Hint: Each of the above functions is easier to implement if you use an auxiliary recursive function*

Part 4 - Submission

- ● Required code organization:
  - ○ program6.c - contains the driver code, and executable program code
  - ○ bst.c/h - Your header file should have (at minimum) the following function declarations:
    - ■ Data struct
      - ● value (int)
    - ■ Node struct
      - ● data (Data)
      - ● left (Node *)
      - ● right (Node *)
    - ■ Tree struct
      - ● root (Node *)
      - ● Data * (*insert)(Tree *, Data);
      - ● Data * (*search)(Tree * bst, Data value);
      - ● void (*sort)(Tree *, Data *);
      - ● int (*compare)(Tree *t, Tree * copy);
      - ● Tree * (*clone)(Tree *t);
      - ● void (*delete)(Tree * bst);
      - ● void (*removeData)(Tree * bst, Data value);
    - ■ Node * newNode(Data d, Node * parent);
    - ■ Tree * newTree();
    - ■ makefile
      - ● *You must have the following labels in your makefile:*
        - ○ all - to compile all your code to an executable called '**program6**' (no extension). **Do not run**.
        - ○ run - to compile if necessary and run
        - ○ checkmem - to compile and run with valgrind
        - ○ clean - to remove all executables and object files
- ● While inside your program 6 folder, create a zip archive with the following command
  - ○ zip -r <yourid>_program6 <files to include>
    - ■ This creates an archive of all file and folders in the current directory called <yourid>_program6.zip
    - ■ **Do not zip the folder itself, only the files required for the program**
- ● Upload the archive to Blackboard under Program 6.