Lab 7 - Binary Search Trees

Due Date: 5:00 p.m., December 21, 2015 (No Grace Period)

- *I strongly recommend to start this lab as early as possible. It is going to be the most difficult lab we do all semester, and you will not be able to finish it in a couple days.*
- *All classes or structs must go into a separate header file with includes guards. Your class methods or related functions must be in a separate source file (.c) and you must use separate compilation in your makefile.*
- *All function interfaces are suggested naming and parameter guidelines. If you feel there is a better way, you are free to alter names, functions interfaces, etc, as long as you follow the lab and style guidelines.*

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 * createTree();
    - Node * createNode();
- Test your functions and structure to ensure everything is initialized correctly by creating a Tree and adding a root to it.

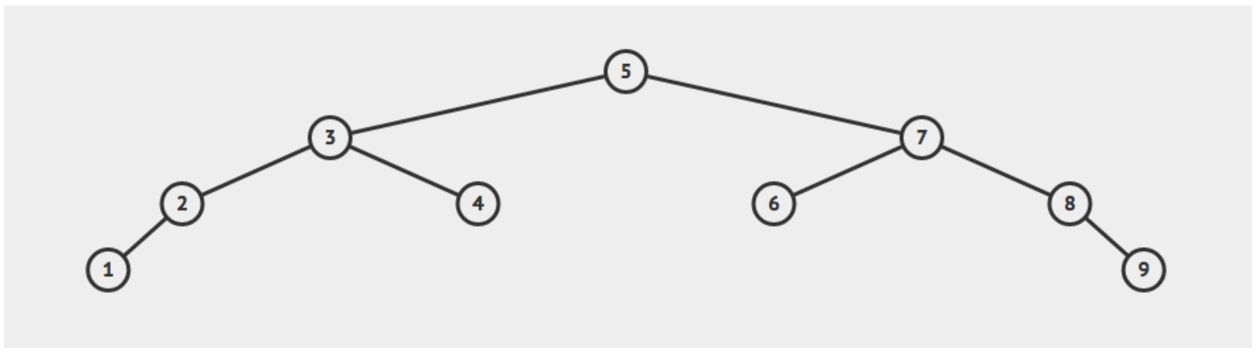
Part 2: BST Operations

- Alter your header file to contain the function declarations for insert, search, remove. Implement the operations in your BST.c file.
- INSERT:
    - Create a function, void insert(Tree * bst, Data value), that inserts into the tree – 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 insert a value into the tree, void insertNode(Node * node, Data value), that you can use for the recursive call

        - If the value is already in the tree, just print a message that you cannot insert duplicate values
- SEARCH:
    - Create a function, Node * search(Tree * bst, Data value), that searches for a value in the tree. You only need to print out if the value is found or not – 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 remove(Tree * bst, Data value), that removes a value from the tree – Helpful hints:
        - Use your (hopefully) working search function to find the node you need to delete
        - 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 with various inputs before starting on the promotion function.

Part 3: Testing Your Tree

- In your main, do the following to test your tree:
    - Using your insert function, read in the 10 integers from attached file, data.txt, create a Data struct for each, and insert them into the tree.
        - Your tree should look like the image below:



    - Using your search function, prompt the user for a value to search for in the tree or 0 to stop searching.
        - Print out the node value, the parent node value, and the child node values, if not a leaf node
    - Using your remove function, prompt the user for a value to remove from the tree or 0 to stop.
        - Print out all values in order from the tree after each remove
- Add a post-order deleteTree() function, then clean up memory by deleting your BST
    - Remember, post order only deletes leafs, so you need only call deleteLeaf()

Part 4: Submission

- Create a tar archive with the command ”tar -cvf lab7.tar.gz .”, and then upload the archive to Blackboard before the deadline. Make sure you do not include the executable in your archive (make clean before creating the archive).
- You will not demo this lab.

Grading Guidelines

- **Part 1:**
    - Part A: 3 points
- **Part 2:**
    - Insert works correctly: 3 points
    - Search works correctly: 2 points
    - Remove: **(5 points)**
        - removes leaf
        - removes node with one branch
        - removes node with two branches
        - removes root node using promotion if there are subtrees
- **Part 3:**
    - Tests by using data.txt: 1 point
- **Style Guidelines and Memory Leaks**
    - Follows Style Guidelines: 1 point
    - Valgrind Shows Memory Leak: -5 points