Lab 2: Binary Search Tree

09-21-2020

My implementations:

* BSTNode.h: I implemented a double-threaded node for the DTBST and getters/setters
* insertHelp(): (Line 67) implemented inside BST.h in the private section
* printHelp(): (Line 141) implemented inside BST.h in the private section
* printInorder(): (Line 220) implemented inside BST.h in the public section
* printReverse(): (Line 224) implemented inside BST.h in the public section

All helper functions used are in the private section of BST.h from lines 34-186.

# Monday 09-21-2020 Pt. 1

## Goals

* Read through .zip contents
* Research bit fields
* Read chapter 5 of the book
* Understand the assignment
* Start to plan my implementation

## Notes

* I am building a double threaded binary tree. This means that leaf nodes will have pointers (threads) that point to the inorder successors and predecessors (left, root, right) of the current node.
* Bitfields allow you to assign variables a memory allocation of a specific number of bits
  + Unsigned int a : 6;
  + For a Boolean bitfield, us bool a : 1;
* I am implementing a DT BST-based dictionary in the BST.h and BSTNode.h files

## Results

* I have a basic understanding of what I am doing
* I think I am going to separate my implementations into separate files so it is clear what work is mine.
* I still want to look at the double-threaded binary search tree implementation linked in the lab more in depth to understand the logic flow

# Monday 09-21-2020 Pt. 2

## Goals

* Now that I have an idea of what I’m gonna be doing, I want to start to build it out.
* I want to build the BSTNode class out to accommodate to the double threading
* I might start to build the BST class, but only enough to test my code in main.cpp.
* I need to make sure to use bitfields and understand how they work
* I don’t understand the point of the dummy node, but I need to look into that

## Notes

* I added the leftBit and rightBit class variables to BSTNode.
* I added four methods to the BSTNode class. They are getters/setters and start on line 45, where it says my name
* I’m stumped as to the need for a dummy pointer. I think I should wait to start writing my class until I clarify its purpose and if I want to use one
* It think it will make it clearer if I move the implementations inside the class, as then you can clearly see what I’ve updated/written

## Results

* I updated the BSTNode class to make it compatible with a double-threaded binary tree. Hopefully the rest of the time I spend programming will be involved the BST class.

# Wednesday 09-23-2020

## Goals

* Read through the linked program to understand the logic
* Implement constructor
* Implement dummy node
* Implement insert function
* Start writing test file

## Notes

* The dummy node provides a place for the left-most and right-most pointers to point to. Instead of keeping these null (which I originally thought might make it easier to traverse) they will be assigned to the dummy root.
* Updated: constructor
* Updated: desctructor
* Built out the inserthelp function
* As I’m following the link to understand the insert logic, it helped to draw a diagram so I can understand what was happening visually
* From my understanding, as you insert you’re tracing down the tree and finding the right spot to insert and then passing the threads to the new node and updating nodes accordingly.

## Results

* I have built the insert method and started to build my main.cpp
* After some testing, the initial insert method I built was causing an infinite loop. I’m glad I started to debugging it.
* After several hours of debugging, I found a simple programmatic mistake. That is fixed so the insert method works (I think)

# Thursday 09-25-2020

## Goals

* Build and test print method
* Understand how traversal works and the logic behind it
* Look at inorder and reverse order printing methods

## Notes

* I started today by drawing a diagram of the tree I’m testing in main.
* Based on my drawing, for the printing of the tree structure I want to go as far right as I can, print everything from the right to the middle element, and then print the left half of the tree.
* I have implemented my algorithm to print the structure.
* I’m having trouble with an infinite loop, that happens when the child points to the parent that points back to the child…
* I need to find a way to see if node has been visited
  + My first thought is to make a variable called previous. This will hold point to the previous node and make sure that if a the algorithm goes up a thread to a parent, it doesn’t get sent right back down to the child that sent it there.
  + That doesn’t work really well, and I’m finding more problems in my algorithm as I continue to add code for edge cases.
* I am going to skip this for now and work on the inorder print method because there’s an example of that in the linked Java doc.
* Well, after implementing the inorder print I don’t know if my insert function is working improperly or if my print method doesn’t work, but something is wrong.
* Fortunately, I just missed going all the way to the left. I’ve since added that while loop. Now it works properly.
* As I’m looking at the output, the printing of the structure is just an inorder printing of the tree but with the keys printed and spaces. That is much simpler than what I was trying to do. Now all I need is a function to get the number of spaces to print.
* I’m gonna do that, and then the reverse order printing should just involve negating the directional elements of the inorder algorithm.

## Results

* All three print functions work as expected and depicted in the output picture from the word doc.
* I have implemented the functions with help from the resources linked in the description.