Emma Zajonc

CSCN 215

Dr. Terri Sipantzi

February 27, 2024

Threaded Binary Tree Approach

I will complete the threaded binary tree in six steps: (1) Add bit fields and the related functions to BSTNode.h, (2) rework the function inserthelp() in BST.h to properly thread each node inserted into the tree, (3) rewrite the function printhelp() to work when there are no longer any null pointers, (4) create the helper functions findNextInOrder() and findPreviousInOrder() to supplement my print functions, (5) create the functions printInOrder() and printReverse() using the helper functions and right and left threads respectively, and finally (6) demonstrate the usage of the binary tree and its functions in main.cpp.

First, I will add bit fields to BSTNode.h. These bit fields will keep track of whether or not the left and right pointers of the node are threaded. Each bit field takes up only one bit of storage, saving space. A value of 0 for the bit field will represent a threaded pointer, while a value of 1 means that the pointer points to a child. I will also add functions that return the left and right bit fields, as well as functions that can be used to set these values.

Second, I will rewrite the function inserthelp(). First, this function must check if the tree is empty, and if it is, it must create the first node in the tree with threaded pointers pointing to itself. If the tree is not empty, inserthelp() must check if the value to be inserted is less than, equal to, or greater than the value of the root. It will continue down the tree recursively in this fashion until it reaches a threaded node, meaning that it has reached the end of the tree. Then, it will add a new node, threading the pointers for the new node appropriately using the information from the previous node.

Third, I will rewrite the printhelp() function. This is not too complicated, but it must be done, because currently the function relies on null pointers to determine when the algorithm has reached the end of a branch. The new printhelp() will rely on BSTNode’s bit fields instead.

Fourth, I will create two helper functions to help print the tree in order and reverse order. The idea for these functions comes from the Java implementation of the binary search tree. These functions, findNextInOrder() and findPreviousInOrder() traverse the tree using threads and find the next position in the tree that should be printed.

Fifth, I will use the two helper functions to implement the new methods printInorder() and printReverse(). These two methods operate in similar ways, first checking if the tree is empty and then using a current pointer to find the leftmost or rightmost position in the tree. From this point, these functions call upon their respective helper functions to follow the threads through the tree, printing the tree’s values in the desired order.

Sixth and finally, I will write a program that tests all of these functions in main.cpp. The program will insert the given set of values into a new binary tree. From there, the program will print the size of the tree, the regular structure of the tree using print(), and two new ordered traversals produced through printInnorder() and printReverse(). This will conclude my work on this project.