Eric Blasko

CSE 330 Data Structures

Winter 2018

Hw #4

**Problem 4.6**

Let:

N = number of nodes

F = number of full nodes

L = number of leaves

H = number of half nodes (nodes with one child)

The total number of nodes in a binary tree is N = F + L + H. When looking at each nodes edges, we find that full nodes have two edges, half nodes have one edge, and leaves have no edges. This gives us 2F + H, which is the total number of edges in a Binary Tree. The number of total edges in a tree is also N – 1. This gives us, 2F + H = N -1 which can be rearranged to H = N – 1 – 2F. Substituting into N = F + H + L, we get N = F + N – 1 – 2F + L. simplifying the equation we end up with F + 1 = L. Thus, the number of full nodes plus one is equal to the number of leaves in a nonempty binary tree.

**Problem 4.31 and 4.32**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Eric Blasko

\* BinaryTree.h

\* 03/12/18

\* This class imitates methods found in BST except that inserttion of data in not ordered. Instead of

\* comparing left and right values, a level order traversal is used to insert a new item. this is done

\* through the use of a queue. This class also has methods to check if it passes for a BST, counts for

\* nodes, leaves, and full nodes

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#ifndef BINARYTREE\_H

#define BINARYTREE\_H

#include <queue>

#include <cassert>

#include <string>

#include <ostream>

#include <limits>

using namespace std;

//Main class

template <typename Comparable>

class BinaryTree

{

public:

BinaryTree(): root{nullptr} {}

//helper methods

void insert(const Comparable & x) {return insert(x,root);}

void insert(Comparable && x) {return insert(move(x),root);}

int nodeCount() {return nodeCount(root);}

int leavesCount() {return leavesCount(root);}

int fullNodeCount() {return fullNodeCount(root);}

void printTree(ostream & out = cout) const;

bool isBST() {return isBST(root);}

private:

struct BinaryNode

{

Comparable element;

BinaryNode \* left;

BinaryNode \* right;

BinaryNode(const Comparable & theElement, BinaryNode \* lt, BinaryNode \* rt):

element{theElement}, left{lt}, right{rt} {}

BinaryNode(Comparable && theElement, BinaryNode \* lt, BinaryNode \* rt):

element{move(theElement)}, left{lt}, right{rt} {}

};

BinaryNode \* root;

void insert(const Comparable &, BinaryNode \* &);

void insert(Comparable &&,BinaryNode \* &);

int nodeCount(BinaryNode \*);

int leavesCount(BinaryNode \*);

int fullNodeCount(BinaryNode \* &);

void printTree(ostream & out, BinaryNode\* t, string indent, const string & tag) const;

bool isBST(BinaryNode \* t);

};

//Copy insert. uses queue to traverse through levels of binary tree to find where to insert

template <typename Comparable>

void BinaryTree<Comparable>::insert(const Comparable & x, BinaryNode \* & t)

{

BinaryNode \* newNode = new BinaryNode(x,nullptr,nullptr);

assert(newNode);

if(t == nullptr)

{

t = newNode;

return;

}

queue<BinaryNode \*> q;

q.push(t);

while(!q.empty())

{

BinaryNode\* temp = q.front();

q.pop();

if(!temp->left)

{

temp->left = newNode;

break;

}

else

q.push(temp->left);

if(!temp->right)

{

temp->right = newNode;

break;

}

else

q.push(temp->right);

}

}

//move insert. uses queue to traverse through levels of binary tre to find where to insert

template <typename Comparable>

void BinaryTree<Comparable>::insert(Comparable && x, BinaryNode \* & t)

{

BinaryNode \* newNode = new BinaryNode(move(x),nullptr,nullptr);

assert(newNode);

if(t == nullptr)

{

t = newNode;

return;

}

queue<BinaryNode \*> q;

q.push(t);

while(!q.empty())

{

BinaryNode\* temp = q.front();

q.pop();

if(!temp->left)

{

temp->left = newNode;

break;

}

else

q.push(temp->left);

if(!temp->right)

{

temp->right = newNode;

break;

}

else

q.push(temp->right);

}

}

//using a queue to traverse through the tree, count is increased before each element is popped

//off of queue

template <typename Comparable>

int BinaryTree<Comparable>::nodeCount( BinaryNode \* t)

{

if(t == nullptr)

return 0;

queue<BinaryNode \*> q;

int count = 0;

q.push(t);

while(!q.empty())

{

count++;

BinaryNode \* temp = q.front();

q.pop();

if(temp->left != nullptr)

q.push(temp->left);

if(temp->right != nullptr)

q.push(temp->right);

}

return count;

}

//using a queue to traverse through the tree, if temp has nullptrs for both left and

//right child, count is increased

template <typename Comparable>

int BinaryTree<Comparable>::leavesCount( BinaryNode \* t)

{

if(t == nullptr)

return 0;

queue<BinaryNode \*> q;

int count = 0;

q.push(t);

while(!q.empty())

{

BinaryNode \* temp = q.front();

q.pop();

if(temp->left == nullptr && temp->right == nullptr)

count++;

if(temp->left != nullptr)

q.push(temp->left);

if(temp->right != nullptr)

q.push(temp->right);

}

return count;

}

//using a queue to traverse through the tree, if left and right are not nullptrs, count

//is increase.

template <typename Comparable>

int BinaryTree<Comparable>::fullNodeCount(BinaryNode \* & t)

{

if(t == nullptr)

return 0;

queue<BinaryNode \*> q;

int count = 0;

q.push(t);

while(!q.empty())

{

BinaryNode \* temp = q.front();

q.pop();

if(temp->left && temp->right)

count++;

if(temp->left != nullptr)

q.push(temp->left);

if(temp->right != nullptr)

q.push(temp->right);

}

return count;

}

//helper method to call print tree

template <typename Comparable>

void BinaryTree<Comparable>::printTree(ostream & out) const

{

cout << "Print Tree\n";

printTree(out,root, "", "");

}

//recursively prints the entire contents of BinaryTree

template <typename Comparable>

void BinaryTree<Comparable>::printTree(ostream & out, BinaryNode \* t, string indent, const string & tag) const

{

if(t == nullptr)

return;

out << indent << tag << t->element << endl;

indent += " ";

printTree(out,t->left, indent, "L ");

printTree(out,t->right, indent, "R ");

}

//using a queue to traverse through tree, if child is not null, compare its element

//to the parent element. left should be less than parent, and right should be greater

//than parent

template <typename Comparable>

bool BinaryTree<Comparable>::isBST(BinaryNode \* t)

{

if(t == nullptr)

return true;

queue<BinaryNode \*> q;

int count = 0;

q.push(t);

while(!q.empty())

{

BinaryNode \* temp = q.front();

q.pop();

if(temp->left != nullptr)

if(temp->left->element > temp->element)

return false;

if(temp->right != nullptr)

if(temp->right->element < temp->element)

return false;

if(temp->left != nullptr)

q.push(temp->left);

if(temp->right != nullptr)

q.push(temp->right);

}

return true;

}

#endif

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Eric Blasko

\* BinaryTree\_test.cpp

\* 03/12/18

\* This program will test the methods in BinaryTree.h. After inserting a new value into

\* the tree, the program will print its count of nodes, leaves, and full nodes. It will

\* also state if it meets requirments for being a BST. Tree will be printed as to have

\* a visual of how the tree looks to compare to the data

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <iostream>

#include "BinaryTree.h"

using namespace std;

//Function to print counts for nodes, leaves, and full nodes

template <typename p>

void print(BinaryTree<p> & t)

{

cout << "# of Nodes: " << t.nodeCount() << " " << endl;

cout << "# of Leaves: " << t.leavesCount() << " " << endl;;

cout << "# of Full Nodes: " <<t.fullNodeCount() << endl;

}

//function to print if tree meets BST standards

template <typename p>

void isBST(BinaryTree<p> & t)

{

if(t.isBST() == true)

cout << "is Binary Search Tree" << endl;

else

cout << "is not Binary Search Tree" << endl;

cout << endl;

}

//Main Function which test functionallty of BinaryTree.h

int main()

{

BinaryTree<int> tree;

cout << "Results while tree is empty\n";

print(tree);

cout << "\n Inserting 19, 3, 20, 1, 4 into tree and with results after each insert\n";

tree.insert(19);

print(tree);

isBST(tree);

tree.insert(3);

print(tree);

isBST(tree);

tree.insert(20);

print(tree);

isBST(tree);

tree.insert(1);

print(tree);

isBST(tree);

tree.insert(4);

print(tree);

isBST(tree);

tree.printTree();

cout << "\nInserting 100, 6, 60, 75, 1, 18, 22 into tree with results after each insert\n";

tree.insert(100);

print(tree);

isBST(tree);

tree.insert(6);

print(tree);

isBST(tree);

tree.insert(60);

print(tree);

isBST(tree);

tree.insert(75);

print(tree);

isBST(tree);

tree.insert(1);

print(tree);

isBST(tree);

tree.insert(18);

print(tree);

isBST(tree);

tree.insert(22);

print(tree);

isBST(tree);

tree.printTree();

}

**Sample Run**

Script started on 2018-03-10 20:46:45-0800

]0;005670557@csusb.edu@csevnc:~/cse330/hw4[005670557@csusb.edu@csevnc hw4]$ g++ -c BinaryTree\_test.cpp

]0;005670557@csusb.edu@csevnc:~/cse330/hw4[005670557@csusb.edu@csevnc hw4]$ g++ BinaryTree\_test.o

]0;005670557@csusb.edu@csevnc:~/cse330/hw4[005670557@csusb.edu@csevnc hw4]$ ./a.out

Results while tree is empty

# of Nodes: 0

# of Leaves: 0

# of Full Nodes: 0

Inserting 19, 3, 20, 1, 4 into tree and with results after each insert

# of Nodes: 1

# of Leaves: 1

# of Full Nodes: 0

is Binary Search Tree

# of Nodes: 2

# of Leaves: 1

# of Full Nodes: 0

is Binary Search Tree

# of Nodes: 3

# of Leaves: 2

# of Full Nodes: 1

is Binary Search Tree

# of Nodes: 4

# of Leaves: 2

# of Full Nodes: 1

is Binary Search Tree

# of Nodes: 5

# of Leaves: 3

# of Full Nodes: 2

is Binary Search Tree

Print Tree

19

L 3

L 1

R 4

R 20

Inserting 100, 6, 60, 75, 1, 18, 22 into tree with results after each insert

# of Nodes: 6

# of Leaves: 3

# of Full Nodes: 2

is not Binary Search Tree

# of Nodes: 7

# of Leaves: 4

# of Full Nodes: 3

is not Binary Search Tree

# of Nodes: 8

# of Leaves: 4

# of Full Nodes: 3

is not Binary Search Tree

# of Nodes: 9

# of Leaves: 5

# of Full Nodes: 4

is not Binary Search Tree

# of Nodes: 10

# of Leaves: 5

# of Full Nodes: 4

is not Binary Search Tree

# of Nodes: 11

# of Leaves: 6

# of Full Nodes: 5

is not Binary Search Tree

# of Nodes: 12

# of Leaves: 6

# of Full Nodes: 5

is not Binary Search Tree

Print Tree

19

L 3

L 1

L 60

R 75

R 4

L 1

R 18

R 20

L 100

L 22

R 6

]0;005670557@csusb.edu@csevnc:~/cse330/hw4[005670557@csusb.edu@csevnc hw4]$ exit

Script done on 2018-03-10 20:47:09-0800