CSCI 255: Lab #5 BST Recursion

Darwin Jacob Groskleg

Tuesday, October 8th, 2019

Contents

Questions

Question 1

What is the output from reverse() with input “CS255”:

- input: “CS255”
- output: “552SC”

Question 2

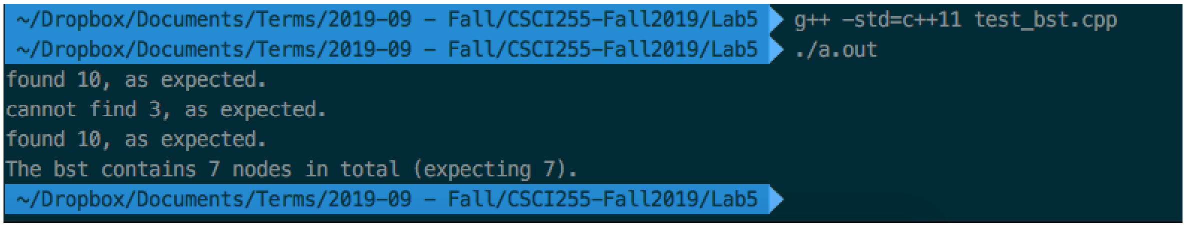
Output from cubes with n=5 : “1 8 27 64 125”.

Recursive definition of cubes:

```
void recursiveCubes(int n) {
    if (n > 0) {
        recursiveCubes(n-1);
        cout << n * n * n << ” ”;
    }
}
```

Question 3

Add the recursive implementations of `count` and `search` on BST. See code on following pages.



```
~/Dropbox/Documents/Terms/2019-09 - Fall/CSCI255-Fall2019/Lab5  g++ -std=c++11 test_bst.cpp
~/Dropbox/Documents/Terms/2019-09 - Fall/CSCI255-Fall2019/Lab5  ./a.out
found 10, as expected.
cannot find 3, as expected.
found 10, as expected.
The bst contains 7 nodes in total (expecting 7).
~/Dropbox/Documents/Terms/2019-09 - Fall/CSCI255-Fall2019/Lab5
```

Figure 1: Console Output

test_bst.cpp

```
/* test_bst.cpp
 * ------------
 * CSCI 255
 * Lab 5:   Recursion, BST
 * Authors: Darwin Jacob Groskleg, Man Lin
 * Date:    Tuesday, October 8, 2019
 *
 * Purpose: Test the probram and make sure that it works correctly.
 *
 * CONSOLE
 * ----------------------------------------------------------------------------
 * $ g++ -std=c++11 test_bst.cpp
 * $ ./a.out
 * found 10, as expected.
 * cannot find 3, as expected.
 * found 10, as expected.
 * The bst contains 7 nodes in total (expecting 7).
 * ----------------------------------------------------------------------------
 */
#include <iostream>
#include ”bst.hpp”

using namespace std;

int main() {
    BST<int> bst;
    bst.insert(5);
    bst.insert(10);
    bst.insert(2);
    bst.insert(15);
    bst.insert(7);
    bst.insert(1);
    bst.insert(4);

    int toSearch;
    BSTNode<int> *result;

    // search 10 by iterative method
    toSearch = 10;
    result = bst.search(toSearch);
    if (result != nullptr)
        cout << ”found ” << result->key << ”, as expected.” << endl;
    else
        cout << ”cannot find ” << toSearch << endl;

    // search 3 by recursive method
    toSearch = 3;
    result = bst.rSearch(toSearch);
    if (result != nullptr)
        cout << ”found ” << result->key << endl;
    else
        cout << ”cannot find ” << toSearch << ”, as expected.” << endl;

    // search 10 by recursive method
    toSearch = 10;
    result = bst.rSearch(toSearch);
    if (result != nullptr)
        cout << ”found ” << result->key << ”, as expected.” << endl;
    else
        cout << ”cannot find ” << toSearch << endl;

    // total # of nodes in bst
    cout << ”The bst contains ” << bst.count() << ” nodes in total”
         << ” (expecting 7).\n”;
}
```

bst.hpp

```
/* bst.hpp
 * --------
 * CSCI 255
 * Lab 5:   Recursion, BST
 * Authors: Darwin Jacob Groskleg, Man Lin
 * Date:    Tuesday, October 8, 2019
 */
#ifndef BST_HPP_INCLUDED
#define BST_HPP_INCLUDED

#if __cplusplus >= 202002L // if c++20
concept LessThanComparable<typename T> {
    bool operator < (const T& lhs, const T& rhs);
}
#endif

// BSTNode class: representing A Node in a Binary Tree
template<class T>
class BSTNode {
public:
    T key;
    BSTNode<T> *left, *right;

    BSTNode() {
        left = right = nullptr;
    }
    BSTNode(const T& el, BSTNode *l = nullptr, BSTNode *r = nullptr) {
        key = el; left = l; right = r;
    }
};

// BST class: Binary Search Tree
//      DJ Groskleg (modified)
template<class T>
    #if __cplusplus >= 202002L // if c++20
        requires LessThanComparable<T>
    #endif
class BST {
    using node = BSTNode<T>;
public:
    BST();

    void insert(const T & el); //insert el to the BST

    //iterative implementation of search
    auto search(const T & el) -> node*;
    //recursive search
    auto rSearch(const T & el) -> node*;

    int count(); //count the number of nodes in the BST

protected:
    node *root;

```

```cpp
private:
    //recursive search helper method: starting from cur BSTNode
    auto recursiveSearch(node *cur, const T & el) -> node*;

    //helper method: count recursively from BSTNode r
    int recursiveCount(node *r);
};


// DJ Groskleg (added)
template <class T>
int BST<T>::recursiveCount(node *r) {
    if (r == nullptr)   return 0;
    return recursiveCount(r->left) + 1 + recursiveCount(r->right);
}

// DJ Groskleg (added)
template <class T>
int BST<T>::count() {
    return recursiveCount(root);
}

// DJ Groskleg (added)
template <class T>
auto BST<T>::rSearch(const T &el) -> node* {
    return recursiveSearch(root, el);
}

// DJ Groskleg (added)
template <class T>
auto BST<T>::recursiveSearch(node *cur, const T &el) -> node* {
    if (cur == nullptr)                           return nullptr;
    if (cur->key == el)                           return cur;

    node* left = recursiveSearch(cur->left, el);
    if (left  != nullptr && left->key  == el)     return left;

    node* right = recursiveSearch(cur->right, el);
    if (right != nullptr && right->key == el)     return right;

    return nullptr;
}

// M.Lin (pre-existing)
template <class T>
BST<T>::BST() {
    root = nullptr;
}

// Insert el to the BST
// M.Lin (pre-existing)
template <class T>
void BST<T>::insert(const T &el) {
    node  *p=nullptr, *prev=nullptr;
    p = root;
    while (p != nullptr){
        prev = p;
        if(p -> key > el)
            p = p-> left;
        else
            p = p->right;
    }
    if (root == nullptr)
        root = new node(el);
    else {
        if (el > prev->key)
            prev->right = new node(el);
        else
            prev->left  = new node(el);
    }
}

// M.Lin (pre-existing)
template <class T>
auto BST<T>::search(const T & el) -> node* {
    node * p = root;
    while (p != 0){
        if (p->key == el)
            return p;
        else{
            if(p-> key < el)
                p = p-> right;
            else
                p = p->left;
        }
    }
    return 0;
}

#endif // BST_HPP_INCLUDED
```