CSCI 255: Lab #5 BST Recursion

Darwin Jacob Groskleg

Tuesday, October 8th, 2019

Contents

Contents																				i
Questions																				1
Question 1																				1
Question 2																				
Question 3	•		 •													•				1
test_bst.cpp																				2
bst.hpp																				4

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 << " ";
   }
}</pre>
```

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

```
1 | /* test_bst.cpp
2
   * CSCI 255
3
   * Lab 5: Recursion, BST
   * Authors: Darwin Jacob Groskleg, Man Lin
               Tuesday, October 8, 2019
   * Date:
7
    * Purpose: Test the probram and make sure that it works correctly.
8
9
   * CONSOLE
10
11
    * $ g++ -std=c++11 test_bst.cpp
12
   * $ ./a.out
13
   * found 10, as expected.
14
   * cannot find 3, as expected.
15
   * found 10, as expected.
   * The bst contains 7 nodes in total (expecting 7).
17
18
19
   */
   #include <iostream>
20
   #include "bst.hpp"
21
22
   using namespace std;
^{23}
24
   int main() {
25
       BST<int> bst;
26
       bst.insert(5);
27
       bst.insert(10);
28
       bst.insert(2);
29
       bst.insert(15);
30
       bst.insert(7);
31
       bst.insert(1);
32
       bst.insert(4);
33
34
       int toSearch;
35
       BSTNode<int> *result;
36
37
       // search 10 by iterative method
38
       toSearch = 10;
39
       result = bst.search(toSearch);
40
       if (result != nullptr)
41
           cout << "found " << result->key << ", as expected." << endl;</pre>
42
       else
43
           cout << "cannot find " << toSearch << endl;</pre>
44
45
       // search 3 by recursive method
46
       toSearch = 3;
47
       result = bst.rSearch(toSearch);
48
       if (result != nullptr)
49
           cout << "found " << result->key << endl;</pre>
50
       else
51
           cout << "cannot find " << toSearch << ", as expected." << endl;</pre>
52
53
       // search 10 by recursive method
54
```

```
toSearch = 10;
55
       result = bst.rSearch(toSearch);
56
       if (result != nullptr)
57
            cout << "found " << result->key << ", as expected." << endl;</pre>
58
       else
59
            cout << "cannot find " << toSearch << endl;</pre>
61
       // total # of nodes in bst
62
       cout << "The bst contains " << bst.count() << " nodes in total"</pre>
63
            << " (expecting 7).\n";
64
65 }
```

bst.hpp

```
1 /* bst.hpp
2
   * CSCI 255
3
   * Lab 5: Recursion, BST
   * Authors: Darwin Jacob Groskleg, Man Lin
               Tuesday, October 8, 2019
   * Date:
  #ifndef BST_HPP_INCLUDED
   #define BST_HPP_INCLUDED
9
10
   #if __cplusplus >= 202002L // if c++20
11
   concept LessThanComparable<typename T> {
12
       bool operator < (const T& lhs, const T& rhs);</pre>
13
14
   #endif
15
16
   // BSTNode class: representing A Node in a Binary Tree
   template<class T>
   class BSTNode {
   public:
20
       T key;
21
       BSTNode<T> *left, *right;
22
^{23}
       BSTNode() {
24
           left = right = nullptr;
25
26
       BSTNode(const T& el, BSTNode *l = nullptr, BSTNode *r = nullptr) {
27
           key = el; left = l; right = r;
28
       }
29
   };
   // BST class: Binary Search Tree
32
          DJ Groskleg (modified)
33
   template<class T>
34
       #if __cplusplus >= 202002L // if c++20
35
           requires LessThanComparable<T>
36
       #endif
37
   class BST {
38
       using node = BSTNode<T>;
39
   public:
40
       BST();
41
42
       void insert(const T & el); //insert el to the BST
43
44
       //iterative implementation of search
45
       auto search(const T & el) -> node*;
46
       //recursive search
47
       auto rSearch(const T & el) -> node*;
48
49
       int count(); //count the number of nodes in the BST
50
   protected:
52
       node *root;
53
54
```

```
private:
55
       //recursive search helper method: starting from cur BSTNode
56
        auto recursiveSearch(node *cur, const T & el) -> node*;
57
58
        //helper method: count recursively from BSTNode r
59
        int recursiveCount(node *r);
60
   };
61
62
63
   // DJ Groskleg (added)
64
   template <class T>
65
   int BST<T>::recursiveCount(node *r) {
        if (r == nullptr) return 0;
67
        return recursiveCount(r->left) + 1 + recursiveCount(r->right);
68
69
70
   // DJ Groskleg (added)
71
   template <class T>
   int BST<T>::count() {
        return recursiveCount(root);
   }
75
76
   // DJ Groskleg (added)
77
   template <class T>
78
   auto BST<T>::rSearch(const T &el) -> node* {
        return recursiveSearch(root, el);
80
81
82
   // DJ Groskleg (added)
83
   template <class T>
84
   auto BST<T>::recursiveSearch(node *cur, const T &el) -> node* {
85
        if (cur == nullptr)
                                                         return nullptr;
86
        if (cur->key == el)
                                                         return cur;
87
88
        node* left = recursiveSearch(cur->left, el);
89
        if (left != nullptr && left->key == el)
                                                         return left;
90
91
        node* right = recursiveSearch(cur->right, el);
92
        if (right != nullptr && right->key == el)
                                                         return right;
93
94
        return nullptr;
95
   }
96
97
   // M.Lin (pre-existing)
98
   template <class T>
   BST<T>::BST() {
100
        root = nullptr;
101
   }
102
103
   // Insert el to the BST
104
   // M.Lin (pre-existing)
105
   template <class T>
106
   void BST<T>::insert(const T &el) {
107
        node *p=nullptr, *prev=nullptr;
108
        p = root;
109
       while (p != nullptr){
110
```

```
prev = p;
111
             if(p \rightarrow key > el)
112
                 p = p-> left;
113
             else
114
                 p = p->right;
115
116
        if (root == nullptr)
117
             root = new node(el);
118
        else {
119
             if (el > prev->key)
120
                  prev->right = new node(el);
121
             else
122
                  prev->left = new node(el);
123
        }
124
    }
125
126
    // M.Lin (pre-existing)
127
    template <class T>
128
    auto BST<T>::search(const T & el) -> node* {
129
        node * p = root;
130
        while (p != 0){
131
             if (p->key == el)
132
                  return p;
133
             else{
134
                  if(p-> key < el)</pre>
135
                      p = p-> right;
136
                  else
137
                      p = p->left;
138
             }
139
        }
140
        return 0;
141
   }
142
#endif // BST_HPP_INCLUDED
```