CSCI 255: Lab #6 BST Traversal

Darwin Jacob Groskleg

Printed: Tuesday, March 3rd, 2020

Contents

Questions & Console Output

```
~/Dropbox/Documents/Terms/2019-09 - Fall/CSCI255-Fall2019/Lab6-BST Traversal make -B a.out

c++  -Iinclude -Isrc -isystem/usr/local/include -isystem/Applications/Xcode.app/Contents/Deve
loper/Toolchains/XcodeDefault.xctoolchain/usr/bin/../include/c++/v1 -isystem/Applications/Xco
de.app/Contents/Developer/Toolchains/XcodeDefault.xctoolchain/usr/bin/../lib/clang/8.0.0/incl
ude -isystem/Applications/Xcode.app/Contents/Developer/Toolchains/XcodeDefault.xctoolchain/us
r/include -isystem/usr/include -std=c++14 -stdlib=libc++ -Wpedantic -Wall -Wextra -g -D_GLIBC
XX_DEBUG -O0  -o main.o -c main.cpp
c++  -Iinclude -Isrc -isystem/usr/local/include -isystem/Applications/Xcode.app/Contents/Deve
loper/Toolchains/XcodeDefault.xctoolchain/usr/bin/../include/c++/v1 -isystem/Applications/Xco
de.app/Contents/Developer/Toolchains/XcodeDefault.xctoolchain/usr/bin/../lib/clang/8.0.0/incl
ude -isystem/Applications/Xcode.app/Contents/Developer/Toolchains/XcodeDefault.xctoolchain/us
r/include -isystem/usr/include -std=c++14 -stdlib=libc++ -Wpedantic -Wall -Wextra -g -D_GLIBC
XX_DEBUG -O0  -o a.out main.o
~/Dropbox/Documents/Terms/2019-09 - Fall/CSCI255-Fall2019/Lab6-BST Traversal ./a.out
Integers to read: 0

Defaulting to saved (unsorted) set of numbers:
        76 28 14 64 90 79 29 33 41 77
Q1. Sorted via Inorder Depth-First Traversal Algorithm:
        14 28 29 33 41 64 76 77 79 90
Q2. Path of the Breath-First Traversal Algorithm:
        76 28 90 14 64 79 29 77 33 41
Compared with Preorder Depth-First Traversal Path:
        76 28 14 64 29 33 41 90 79 77
~/Dropbox/Documents/Terms/2019-09 - Fall/CSCI255-Fall2019/Lab6-BST Traversal ./a.out
Integers to read: 5
>       3 5 9 0 2

Q1. Sorted via Inorder Depth-First Traversal Algorithm:
        0 2 3 5 9
Q2. Path of the Breath-First Traversal Algorithm:
        3 0 5 2 9
Compared with Preorder Depth-First Traversal Path:
        3 0 2 5 9
~/Dropbox/Documents/Terms/2019-09 - Fall/CSCI255-Fall2019/Lab6-BST Traversal
```

Figure 1: Compiles and runs.

main.cpp

```cpp
/* main.cpp (originally TestBST.cpp)
 * ---------------------------------
 * CSCI 255
 * Lab 6:   BST Traversal
 * Authors: Darwin Jacob Groskleg
 * Purpose: to test TraversableBST's capacity to be used to implement a sorting
 *          algorithm of numbers.
 */
#include <iostream>
#include <array>
#include "traversable_bst.hpp"
using namespace std;

int main() {
// 1. In your main.cpp file, implement a sorting algorithm based on inorder
//    traversal of BST:
    // 1. Read n integers. 2. Create an empty BST.
    // 3. Insert the n integers into the BST by repeatedly using the insert
    //    method that you already implemented in the previous lab.
    TraversableBST<int> bst;
    cerr << "Integers to read: ";
    int n = 0;
    cin >> n;
    if (n > 0) {
        cerr << ">\t";
        int z;
        while (n--) {
            cin >> z;
            bst.insert(z);
        }
    } else {    // default case
        cout << "\nDefaulting to saved (unsorted) set of numbers: \n\t";
        array<int,10> arbit_z_10{{ 76, 28, 14, 64, 90, 79, 29, 33, 41, 77 }};
        for (auto v : arbit_z_10) {
            cout << v << ' ';
            bst.insert(v);
        }
    }
    // Depth-First Traversal Sorting
    // 4. Call the inorder method to do an inorder traversal of the BST and
    //    print the n integers. The numbers will be printed in ascending sorted
    //    order.
    cout << "\nQ1. Sorted via Inorder Depth-First Traversal Algorithm:\n\t";
    bst.inorder();

// 2. Implement the breath-first traversal algorithm.
    cout << "\nQ2. Path of the Breath-First Traversal Algorithm:\n\t";
    bst.breadthFirst();

    cout << "\nCompared with Preorder Depth-First Traversal Path:\n\t";
    bst.preorder();

    return 0;
}
```

traversable_bst.hpp

```cpp
/* traversable_bst.hpp
 * -------------------
 * CSCI 255
 * Lab 6:   BST Traversal
 * Authors: Darwin Jacob Groskleg
 *
 * Purpose: to extend the BST class from Lab #5 to be able to do breadth-first
 *          traversal operations.
 *
 * Open-Closed Principle: be open to extension, closed for modification.
 */
#ifndef TRAVERSABLE_BST_HPP_INCLUDED
#define TRAVERSABLE_BST_HPP_INCLUDED

#include "bst.hpp"

#include <initializer_list>
#include <queue>
#include <iostream>

template <typename T>
class TraversableBST : public BST<T> {
    using node = BSTNode<T>;
public:
    TraversableBST() = default; // required for inheriting class
    // Copy-Constructor is easy when you can traverse the copy
    TraversableBST(const TraversableBST& rhs) {
        auto ordered_values = rhs.breadthFirstCopy(rhs->root);
        for (auto &v :ordered_values)
            this->insert(v);
    }
    TraversableBST(std::initializer_list<T> inorder_list) {
        for (auto &n : inorder_list)
            this->insert(n);
    }

// Depth-First Traversal
    // Sorted Order
    void inorder(node* p);
    inline void inorder() { inorder(this->root); }

    // By-Row Starting from Root
    void preorder(node* p);
    inline void preorder() { preorder(this->root); }

// Breadth-First Traversal
    void breadthFirst(node* p);
    inline void breadthFirst() { breadthFirst(this->root); }

private:
    std::queue<T> breadthFirstCopy(node* p) const;
    inline void visit(node* p) { std::cout << p->key << ' '; }
};

```

```
55
56  /// IMPLEMENTATION
57
58  template<typename T>
59  void TraversableBST<T>::inorder(node* p) {
60      if (p != nullptr) {
61          inorder(p->left);
62          visit(p);
63          inorder(p->right);
64      }
65  }
66
67  template<typename T>
68  void TraversableBST<T>::preorder(node* p) {
69      if (p != nullptr) {
70          visit(p);
71          preorder(p->left);
72          preorder(p->right);
73      }
74  }
75
76  template<typename T>
77  void TraversableBST<T>::breadthFirst(node* p) {
78      for (auto values = breadthFirstCopy(p); !values.empty(); values.pop())
79          std::cout << values.front() << ' ';
80  }
81
82  template<typename T>
83  std::queue<T> TraversableBST<T>::breadthFirstCopy(node* p) const {
84      std::queue<node*> frontier;
85      std::queue<T> pipe;
86
87      if (p != nullptr) {
88          frontier.push(p);
89          while (!frontier.empty()) {
90              p = frontier.front();
91              frontier.pop();
92
93              pipe.push(p->key);
94
95              if (p->left != nullptr)
96                  frontier.push(p->left);
97              if (p->right != nullptr)
98                  frontier.push(p->right);
99          }
100     }
101     return pipe;
102 }
103
104 #endif // TRAVERSABLE_BST_HPP_INCLUDED
```

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;

```

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

```
111         prev = p;
112         if(p -> key > el)
113             p = p-> left;
114         else
115             p = p->right;
116     }
117     if (root == nullptr)
118         root = new node(el);
119     else {
120         if (el > prev->key)
121             prev->right = new node(el);
122         else
123             prev->left  = new node(el);
124     }
125 }
126 
127 // M.Lin (pre-existing)
128 template <class T>
129 auto BST<T>::search(const T & el) -> node* {
130     node * p = root;
131     while (p != 0){
132         if (p->key == el)
133             return p;
134         else{
135             if(p-> key < el)
136                 p = p-> right;
137             else
138                 p = p->left;
139         }
140     }
141     return 0;
142 }
143 
144 #endif // BST_HPP_INCLUDED
```