20/20

Printout for cs320-09-A21-LLRBTree.hpp

```
// File: LLRBTree/LLRBTree.hpp
// Olivia Lara
// November 22, 2017

#ifndef LLRBTREE_HPP_
#define LLRBTREE_HPP_

#include <iostream> // ostream.
using namespace std;

template<class T> class Node; // Forward declaration.

const bool RED = true;
const bool BLACK = false;

template<class T>
class LLRBTree {
private:
    Node<T> *_root;

public:
    LLRBTree(); //Constructor
    // Post: This left-leaning red-black tree is initialized to be empty.

    ~LLRBTree(); //Destructor
    // Post: This left-leaning red-black tree is deallocated.

    bool contains(T const &data);
    // Post: Returns true if this tree contains data.

    void insert(T const &data);
    // Post: data is stored in this tree in order with no duplicates.

    void remove(T const &data);
    // Post: If data is found, then it is removed from this tree.

    void toStream(ostream &os) const;
    // Post: A string representation of this tree is streamed to os.
};

template<class T>
class Node {
    friend class LLRBTree<T>;

private:
    Node *_left;
    T _data;
    Node *_right;
    bool _color; // Color of link from parent.

public:
    Node(T data);
    // Post: This node is allocated with _data set to data, _color
    // set to RED, and _left, _right set to nullptr.

    ~Node();
    // Post: This node, its left subtree, and its right subtree are deallocated.

private:
    Node<T> *fixup(Node<T> *h);
    // Pre: h->_left and h->_right are roots of left-leaning red-black trees.
    // Post: h is a root of a left-leaning red-black tree.
    // Post: A pointer to the root node of the modified tree is returned.
    void flipColors(Node<T> *h);
    // Pre: h has nonempty _left and _right.
    // Post: The colors of h and its children are flipped.

    Node<T> *insert(Node<T> *h, T const &data);
    // Post: val is stored in root node h in order as a leaf.
    // Post: A pointer to the root node of the modified tree is returned.

    bool isRed(Node<T> *h);
    // Post: Returns false if h == nullptr.
    // Otherwise returns true if h node is RED.

    T min(Node<T> *h);
    // Pre: h != nullptr.
    // Post: The minimum of tree rooted at h is returned.

    Node<T> *moveRedLeft(Node<T> *h);
    // Pre: h has nonempty _left and _right.
    // Invariant: Either h or h->_left is RED.
    // Post: A pointer to the root node of the modified tree is returned.

    Node<T> *moveRedRight(Node<T> *h);
    // Pre: h has nonempty _left and _right.
    // Invariant: Either h or h->_right is RED.
    // Post: A pointer to the root node of the modified tree is returned.

    Node<T> *remove(Node<T> *h, T const &data);
    // Post: data is removed from root node h.
    // Post: A pointer to the root node of the modified tree is returned.

    Node<T> *removeMin(Node<T> *h);
    // Pre: h != nullptr.
    // Post: The minimum value is removed from root node h.
    // Post: A pointer to the root node of the modified tree is returned.

    Node<T> *rotateLeft(Node<T> *h);
    // Pre: h->_right is RED.
    // Post: h is rotated left.
    // Post: A pointer p to the root node of the modified tree is returned.
    // Post: p->left is RED.
    // Unchanged: Color of link to new root from its parent.

    Node<T> *rotateRight(Node<T> *h);
    // Pre: h->_left is RED.
    // Post: h is rotated right.
    // Post: A pointer p to the root node of the modified tree is returned.
    // Post: p->right is RED.
    // Unchanged: Color of link to new root from its parent.

    void toStream(Node<T> *h, string prRight, string prRoot, string prLeft, ostream &os)
const;
};

// ========= Constructors =========
template<class T>
LLRBTree<T>::LLRBTree() {
    _root = nullptr; // The empty tree.
}

template<class T>
Node<T>::Node(T data) :
    _color(RED),
    _data(data),
    _left(nullptr),
    _right(nullptr) {
    }
```

```
// ========= Destructors =========
template<class T>
LLRBTree<T>::~LLRBTree() {
    if (_root != nullptr) {
        delete _root;
        _root = nullptr;
    }
}

template<class T>
Node<T>::~Node() {
    if (_left != nullptr) {
        delete _left; // Delete left subtree.
        _left = nullptr;
    }
    if (_right != nullptr) {
        delete _right; // Delete right subtree.
        _right = nullptr;
    }
}

// ========= contains =========
template<class T>
bool LLRBTree<T>::contains(T const &data) {
    Node<T> *p = _root;
    while (p != nullptr) {
        if (p->_data > data) {
            p = p-> _left;
        } else if (p->_data < data) {
            p = p-> _right;
        } else if (p->_data == data) {
            return true;
        }
    }
    return false;
}

// ========= fixup =========
template<class T>
Node<T> *Node<T>::fixup(Node<T> *h) {
    if (isRed(h->_right)) {
        h = rotateLeft(h);
        // cout << "1 - fixup() rotateLeft" << endl;
    }
    if (isRed(h->_left) && isRed(h->_left->_left)) {
        h = rotateRight(h);
        // cout << "2 - fixup() rotateRight" << endl;
    }
    if (isRed(h->_left) && isRed(h->_right)) {
        flipColors(h);
        // cout << "3 - fixup() flipColors" << endl;
    }
    return h;
}

// ========= flipColors =========
template<class T>
void Node<T>::flipColors(Node<T> *h) {
    h->_color = !h->_color;
    h->_left->_color = !h->_left->_color;
    h->_right->_color = !h->_right->_color;
}

// ========= insert =========
template<class T>
```

```
void LLRBTree<T>::insert(T const &data) {
    _root = _root->insert(_root, data);
    _root->_color = BLACK;
}

template<class T>
Node<T> *Node<T>::insert(Node<T> *h, T const &data) {
    if (h == nullptr) {
        return new Node<T>(data);
    }
    if (data < h->_data) {
        h->_left = insert(h->_left, data);
    }
    else if (data > h->_data) {
        h->_right = insert(h->_right, data);
    } // else no duplicates.
    return fixup(h);
}

// ========= isRed =========
template<class T>
bool Node<T>::isRed(Node<T> *h) {
    return h == nullptr ? BLACK : h->_color;
}

// ========= min =========
template<class T>
T Node<T>::min(Node<T> *h) {
    Node<T> *p = h->_left;
    if(p == nullptr){
        return h->_data;
    }
    while (p->_left != nullptr) {
        p = p->_left;
    }
    return p->_data;
}

// ========= moveRedLeft =========
template<class T>
Node<T> *Node<T>::moveRedLeft(Node<T> *h) {
    flipColors(h);
    if (isRed(h->_right->_left)) {
        h->_right = rotateRight(h->_right);
        h = rotateLeft(h);
        flipColors(h);
        // cout << "4 - moveRedLeft()" << endl;
    }
    return h;
}

// ========= moveRedRight =========
template<class T>
Node<T> *Node<T>::moveRedRight(Node<T> *h) {
    flipColors(h);
    if (isRed(h->_left->_left)) {
        h = rotateRight(h);
        flipColors(h);
        // cout << "5 - moveRedRight" << endl;
    }
    return h;
}

// ========= remove =========
template<class T>
void LLRBTree<T>::remove(T const &data) {
    if (!contains(data)) {
        return;
    }
    _root = _root->remove(_root, data);
    if (_root != nullptr) {
        _root->_color = BLACK;
    }
}

template<class T>
Node<T> *Node<T>::remove(Node<T> *h, T const &data) {
    if (data < h->_data) {
        if (!isRed(h->_left) && !isRed(h->_left->_left)) {
            h = moveRedLeft(h);
            // cout << "8 - remove(), left and left->left black" << endl;
        }
        h->_left = remove(h->_left, data);
    }
    else {
        if (isRed(h->_left)) {
            h = rotateRight(h);
            // cout << "9 - remove(), left red" << endl;
        }
        if ((data == h->_data) && (h->_right == nullptr)) {
            // cout << "10 - remove(), data match and no right" << endl;
            delete h;
            return nullptr;
        }
        if (!isRed(h->_right) && !isRed(h->_right->_left)) {
            h = moveRedRight(h);
            // cout << "11 - remove(), right and right->left black" << endl;
        }
        if (data == h->_data) {
            h->_data = min(h->_right);
            h->_right = removeMin(h->_right);
            // cout << "12 - remove(), data match" << endl;
        }
        else {
            h->_right = remove(h->_right, data);
            // cout << "13 - remove(), removed right" << endl;
        }
    }
    return fixup(h);
}

// ========= removeMin =========
template<class T>
Node<T> *Node<T>::removeMin(Node<T> *h) {
    if (h->_left == nullptr) {
        delete h;
        // cout << "6 - removeMin(), no left" << endl;
        return nullptr;
    }
    if (!isRed(h->_left) && !isRed(h->_left->_left)) {
        h = moveRedLeft(h);
        // cout << "7 - removeMin(), left and left->left black" << endl;
    }
    h->_left = removeMin(h->_left);
    return fixup(h);
}

// ========= rotateLeft =========
template<class T>
Node<T> *Node<T>::rotateLeft(Node<T> *h) {
    Node<T> *p = h->_right;
    h->_right = p->_left;
    p->_left = h;
    p->_color = h->_color;
    h->_color = RED;
    return p;
}

// ========= rotateRight =========
template<class T>
Node<T> *Node<T>::rotateRight(Node<T> *h) {
    Node<T> *p = h->_left;
    h->_left = p->_right;
    p->_right = h;
    p->_color = h->_color;
    h->_color = RED;
    return p;
}

// ========= operator<< =========
template<class T >
ostream & operator<<(ostream &os, const LLRBTree<T> &rhs) {
    rhs.toStream(os);
    return os;
}

// ========= toStream =========
char toChar(bool color) {
    return color ? 'r' : 'b';
}

template<class T>
void LLRBTree<T>::toStream(ostream & os) const {
    if (_root == nullptr) {
        os << "(b)";
    }
    else {
        _root->toStream(_root, "", "", "", os);
    }
}

template<class T>
void Node<T>::toStream(Node<T> *h, string prRight, string prRoot, string prLeft, ostream
& os) const {
    if (h->_right == nullptr) {
        os << prRight << "      -(b)" << endl;
    }
    else {
        toStream(h->_right, prRight + "      ", prRight + "      ", prRight + "     |",
os);
    }
    os << prRoot << toChar(h->_color);
    os.fill('-');
    os.width(4);
    os.setf(ios::left, ios::adjustfield);
    os << h->_data << "|" << endl;
    if (h->_left == nullptr) {
        os << prLeft << "      -(b)" << endl;
    }
    else {
        toStream(h->_left, prLeft + "     |", prLeft + "      ", prLeft + "      ", os);
    }
}

#endif

// new page
// new page
```

```
clang++ -stdlib=libc++ -std=c++11 -c LLRBTreeMain.cpp \
        -I ../dp4dsDistribution/Utilities 2>&1
clang++ -stdlib=libc++ -std=c++11 -o LLRBTreeMain LLRBTreeMain.o \
        ../dp4dsDistribution/Utilities/Utilities.o
strip LLRBTreeMain

// new page
```

```
Testing cs320-09

Testing cs320-09 unit-contains
==================================

            -(b)
      b60--|
     |     |     -(b)
     |      r50--|
     |           -(b)
b40--|
     |           -(b)
     |      b30--|
     |     |     -(b)
      r20--|
           |     -(b)
            b10--|
                 -(b)

The tree contains 10.

The tree contains 20.

The tree contains 30.

The tree contains 40.

The tree contains 50.

The tree contains 60.

The tree does not contain 5.

The tree does not contain 15.

The tree does not contain 25.

The tree does not contain 35.

The tree does not contain 45.

The tree does not contain 55.

The tree does not contain 65.

// new page
```

```
Testing cs320-09 unit-insert
================================

            -(b)
      b60--|
     |     |      -(b)
     |      r50--|
     |            -(b)
b40--|
     |            -(b)
     |      b30--|
     |     |      -(b)
      r20--|
           |      -(b)
            b10--|
                  -(b)

// new page
```

```
Testing cs320-09 unit-remove
================================

                  -(b)
            b90--|
            |     -(b)
      b80--|
      |     |           -(b)
      |     |     b70--|
      |     |     |     -(b)
      |     r60--|
      |           |     -(b)
      |           b50--|
      |                 -(b)
b40--|
      |           -(b)
      |     b30--|
      |     |     -(b)
      b20--|
            |     -(b)
            b10--|
                  -(b)

                  -(b)
            b90--|
            |     -(b)
      b80--|
      |     |     -(b)
      |     b70--|
      |           -(b)
b60--|
      |           -(b)
      |     b50--|
      |     |     -(b)
      b40--|
            |     -(b)
            b30--|
                  |     -(b)
                  r10--|
                        -(b)

                  -(b)
            b90--|
            |     -(b)
      b80--|
      |     |     -(b)
      |     b70--|
      |           -(b)
b60--|
      |           -(b)
      |     b40--|
      |     |     -(b)
      b30--|
            |     -(b)
            b10--|
                  -(b)

            -(b)
      b90--|
      |     |     -(b)
      |     r80--|
      |           -(b)
b70--|
      |           -(b)
      |     b40--|
        |       |       -(b)
         r30--|
                |       -(b)
                 b10--|
                        -(b)

                 -(b)
         b90--|
        |        -(b)
b80--|
        |               -(b)
        |        b40--|
        |       |       -(b)
         r30--|
                |       -(b)
                 b10--|
                        -(b)

// new page
```