Anthony Esmeralda

Kevin Ngo

Professor Vo

CSCI 220

December 4, 2017

Project 4: Binary Search Tree

Compiler: Microsoft Visual Studio 2017

Language: C++

Files Used:

p4large.txt

Entry.h

BinaryTree.h

SearchTree.h

AVLTree.h

BeginProgram.h

Entry.cpp

BinaryTree.cpp

SearchTree.cpp

AVLTree.cpp

BeginProgram.cpp

Main.cpp

Kevin’s Notes

This project is fully functional. The most troubling part for us was the restructuring of the AVL Tree. We felt like this took most of the time because most of the other code was given to us through the book and we just had to convert it to be a non-template tree. While converting the template to non-template tree, there were many problems that arose, such as how we would split up the classes, or which parameters we would have to pass. These were mainly just design issues that we faced but we ended up deciding to make the program much more modulable in case we would ever need it in the future. Personally, for me, (Kevin), I found the book’s code extremely hard to follow at certain points. I had done most of my work in Java for the semester, so I needed to review the C++ syntax understand the code. On top of that, the code seemed to be very hard to follow even after I had reviewed some syntax because it required a solid foundation in the data structures we were learning in class to comprehend it. Other than that, it was not too difficult of a project. We (Anthony and I) split up the work and were both able to benefit from this project. This project helped us to become much more familiar with the debugger as well as the concept of the AVL Tree and the Binary Search Tree. On a side note, since this was a group project, Anthony and I also used this to our advantage and forced ourselves to learn how to use Git Repositories. We did this so we could easily share and update code as well as take advantage of version control. In addition, by using Git, we also got more familiar with the command line and the bash/terminal window, which is present in macOS and Linux. Learning this was one of the most important keys for this project. It allowed us to update and share our work almost instantaneously with one another as well as stay in good communication.

Anthony’s Notes

Like Kevin said, this program is fully functional. The most troubling part of the program was doing the code for the restructure, setting up functions for the Search tree class to work with a specific data type, and the positions class (which was the inner class of the binary tree). I believed that it was best to pass the Entry class object that held all our data (county state code, population, and county state name) within our public functions of the Search tree and AVL tree so that our parameters wouldn’t have 4 things you would need to pass in and that it was just easier to keep track of what we were passing in to fill in our tree or erase data. In addition, when creating the code for restructure and building the AVL tree I noticed that with the given Binary Tree class, that there wasn’t enough data or functions for it to fully work so we had to implement our own such as having the position class have functions and data that can take store the height, setting the parents of this node, setting the right and left children of a node, and in also setting up a new root if needed if a restructure happened. Besides from the technical stuff, my experience with this team project was pretty good. We tried new things out such as using GitHub’s repository so that we can always be up to date with our code if someone were to make any changes and it made communications a lot easier between the two of us.

Comparison of AVL vs BST

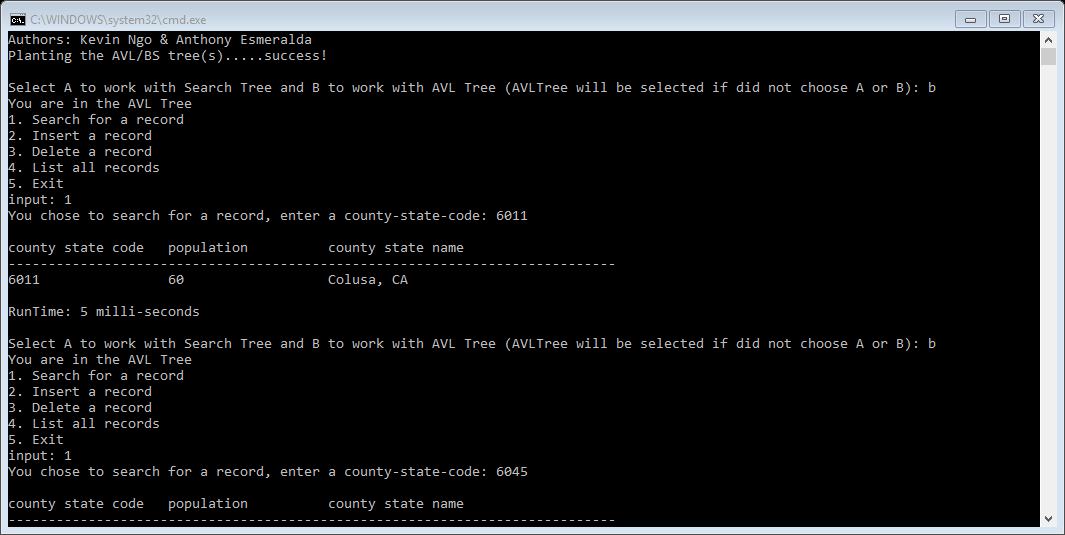
|  |  |  |  |
| --- | --- | --- | --- |
| Key | Function | AVL Runtime | BST Runtime |
| 6011 | Search | 5 milliseconds | 8 milliseconds |
| 6045 | Search | 5 milliseconds | 7 milliseconds |
| 6103 | Search | 6 milliseconds | 7 milliseconds |
| 6004 | Insert | 6 milliseconds | 4 milliseconds |
| 6060 | Insert | 6 milliseconds | 8 milliseconds |
| 6078 | Insert | 7 milliseconds | 9 milliseconds |
| 6061 | Erase | 6 milliseconds | 6 milliseconds |
| 6045 | Erase | 5 milliseconds | 7 milliseconds |
| 6049 | Erase | 1 milliseconds | 2 milliseconds |
| 6113 | Search | 4 milliseconds | 8 milliseconds |
| 8115 | Insert | 6 milliseconds | 8 milliseconds |
| 6089 | Erase | 6 milliseconds | 5 milliseconds |

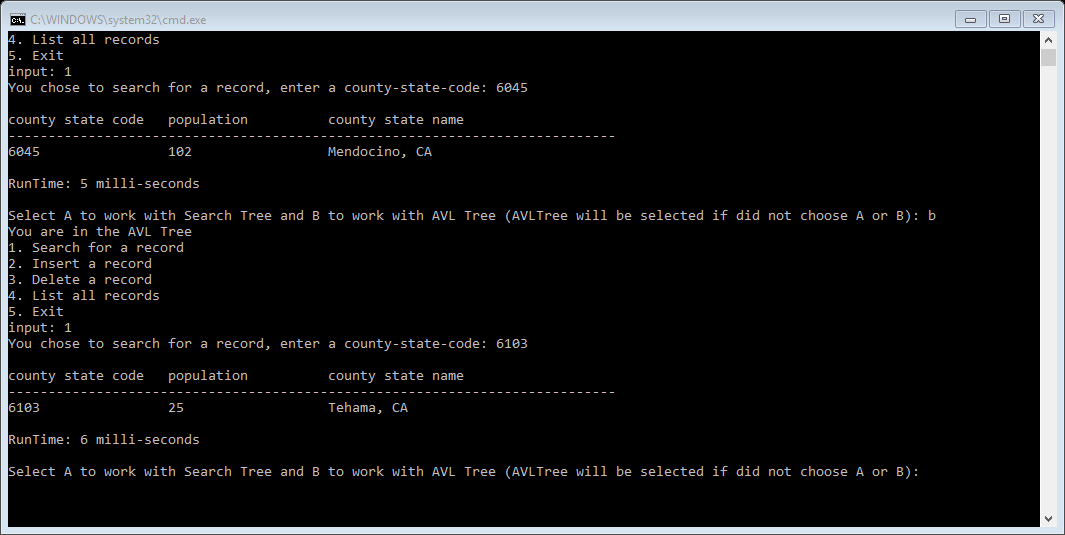
Notes on differences

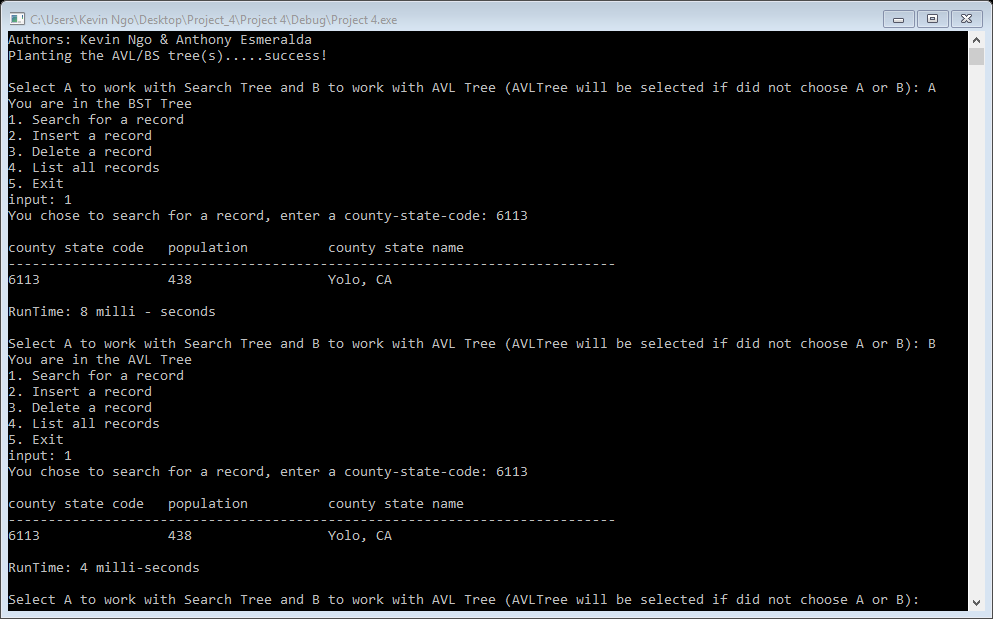
Based on the data documented, it is obvious the AVL Tree seems to be more efficient than the BST, on average. As it can be seen, on our very last case, there was an erase that was faster in the BST than it was in the AVL Tree. There are differences like this, most likely do to the restructuring required after performing such actions. Although the cases for erase and insert vary (due to restructuring), it is obvious to see that the search is much more efficient than the BST. In some cases, our AVL searches were better than the BST searches (in terms of milliseconds) by a factor or two. This proves that although the AVL takes some more time to insert and also remove (due to the restructure requirement), when searching the AVL Tree is more efficient than the BST tree and is faster at locating records.

Input/Output

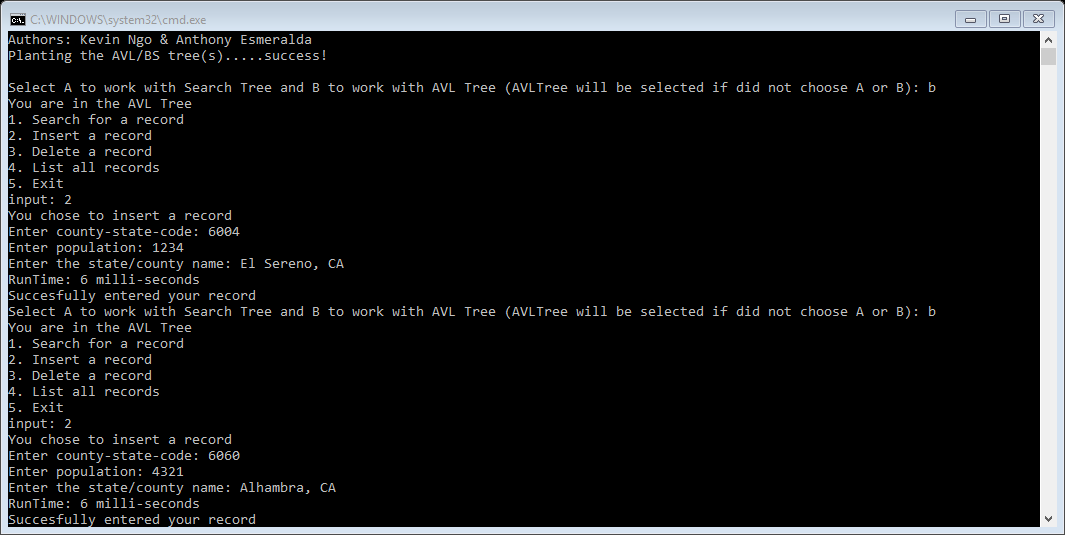
Option 1:

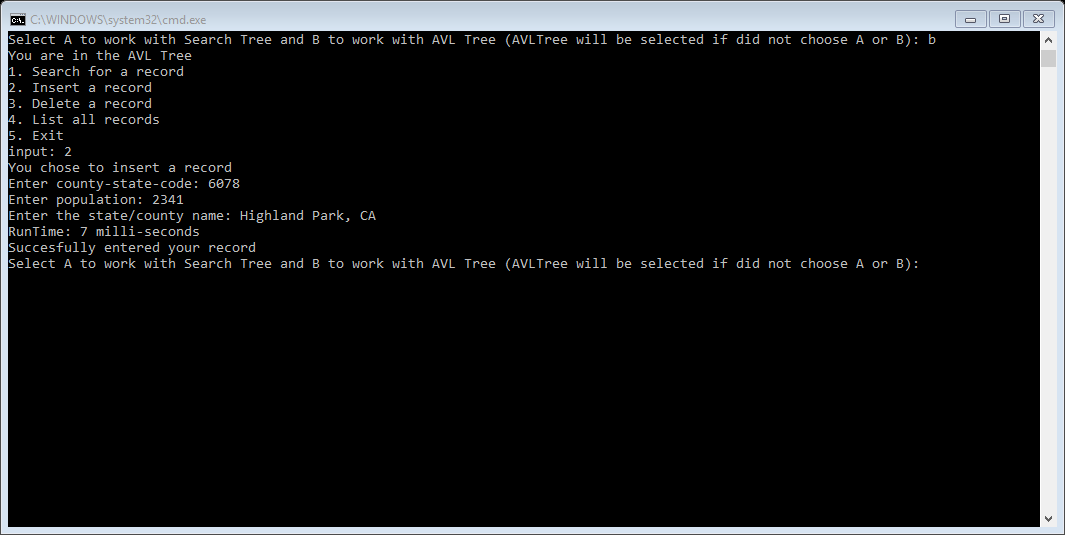


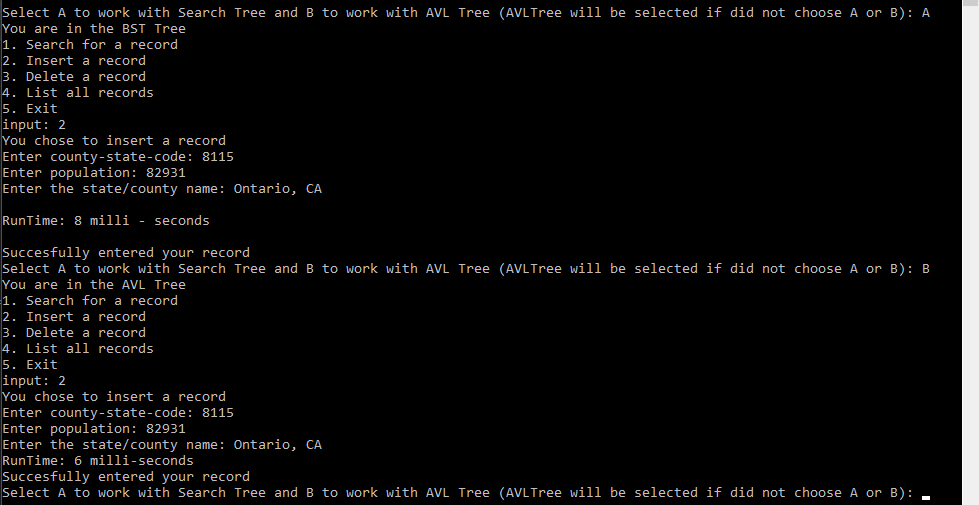




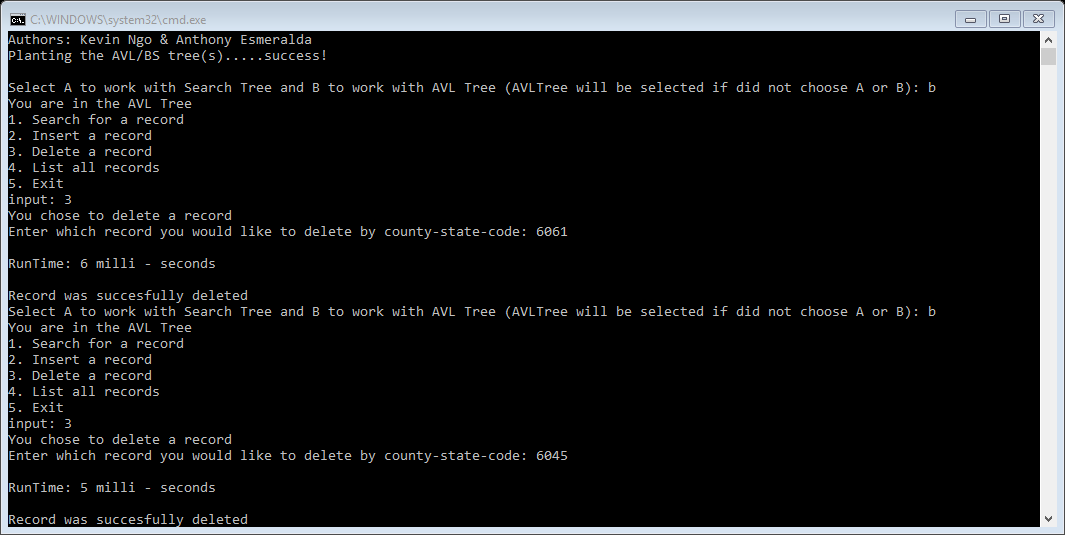
Option 2:

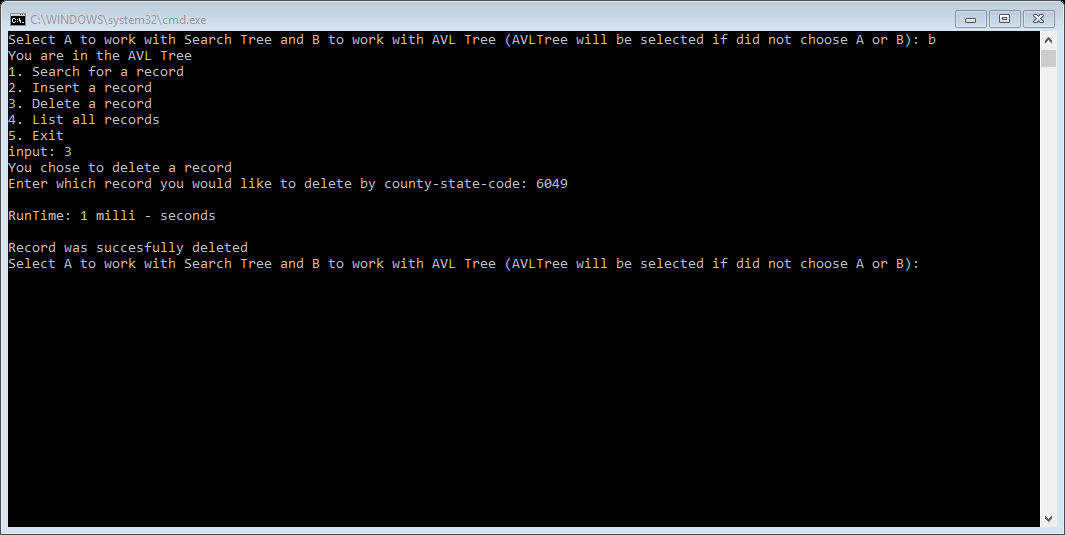


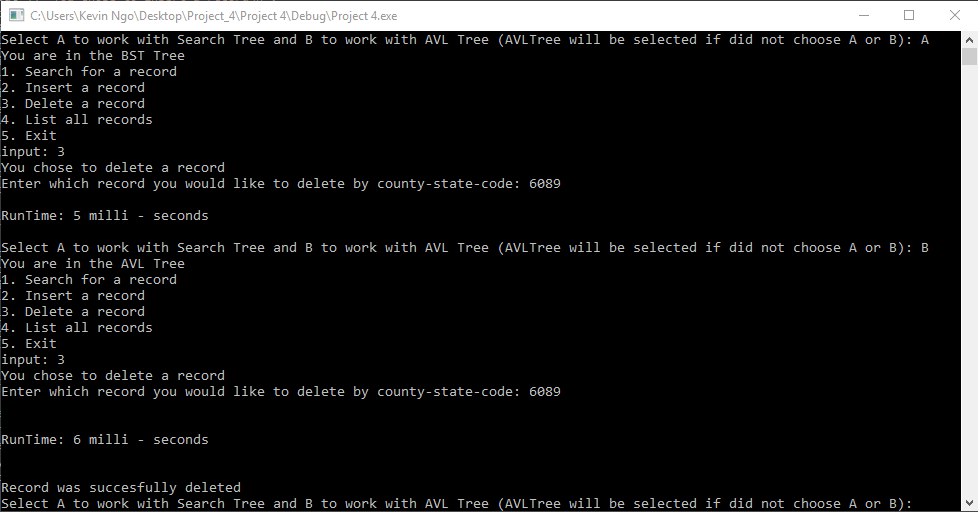




Option 3:

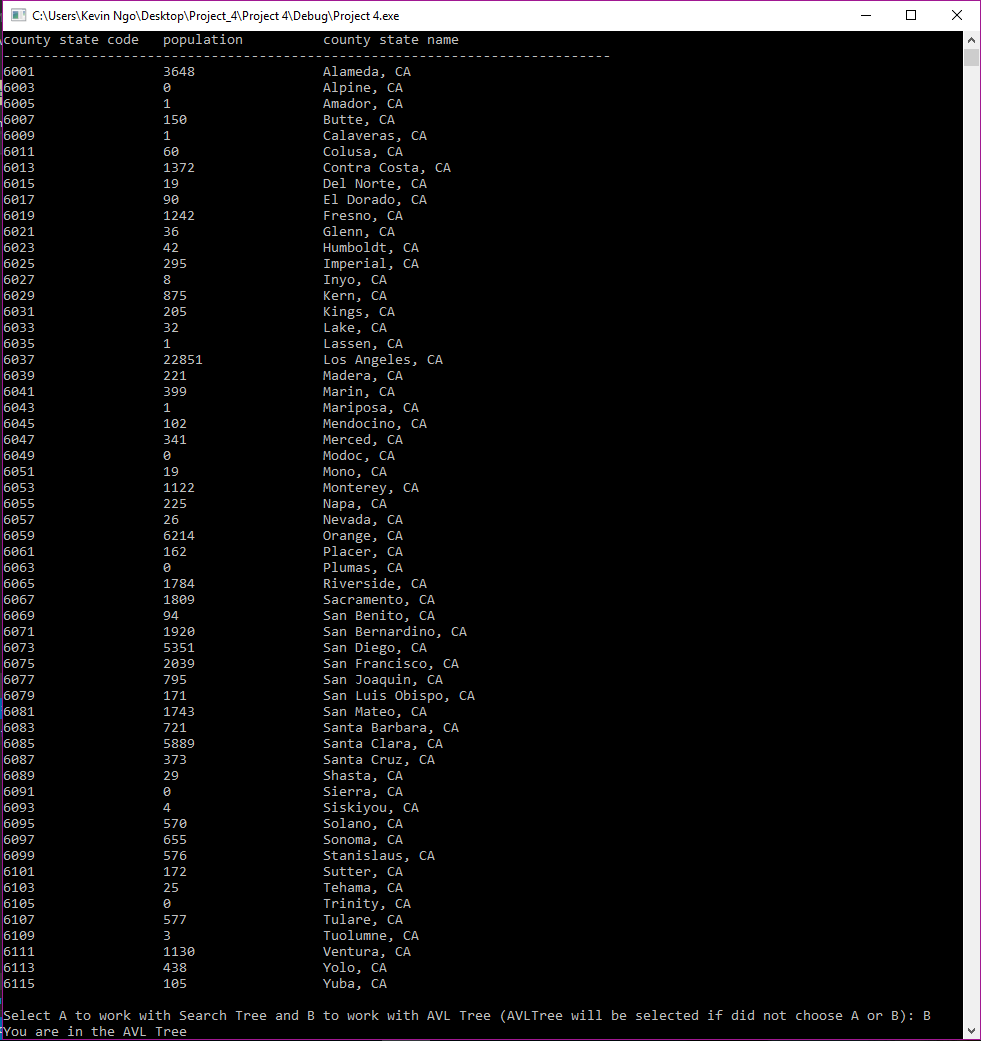




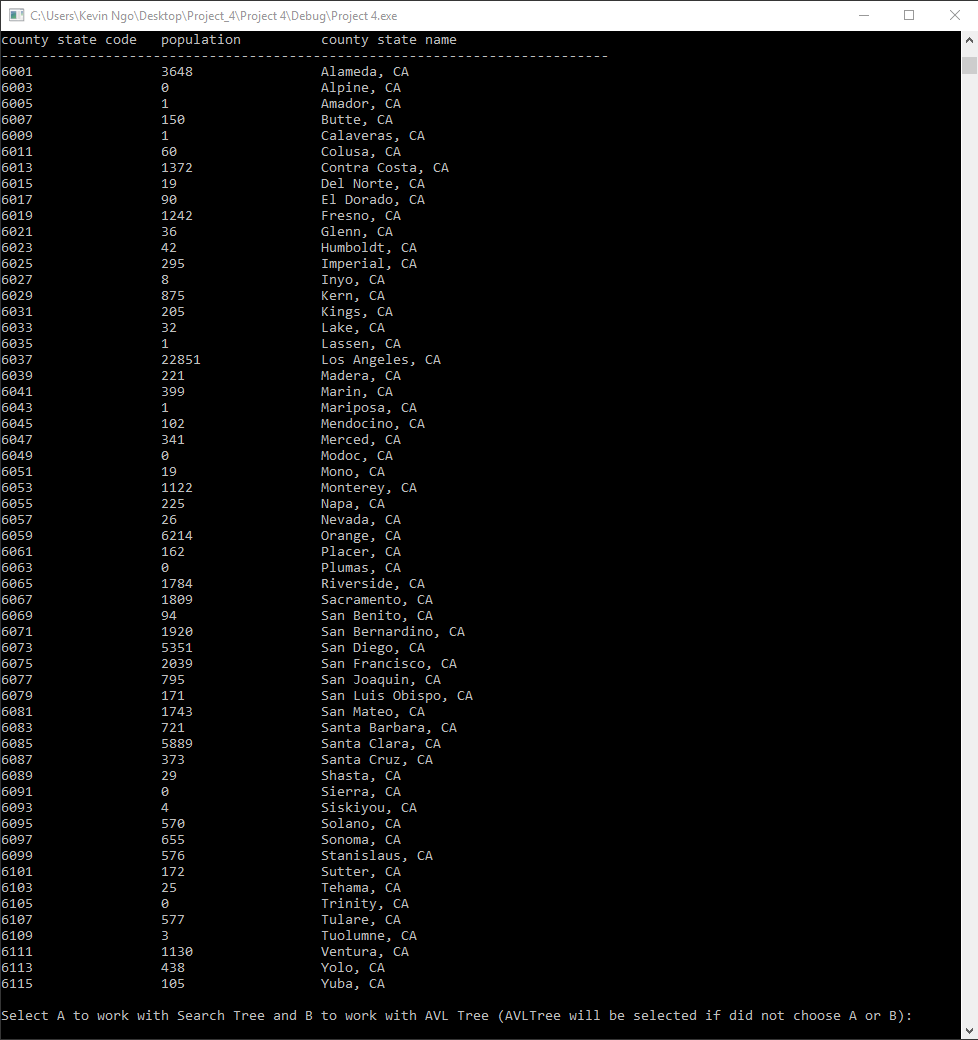


Option 4:

(BST)



(AVL)



Source Code

Headers:

AVLTree.h

/\* Program: Project 4 - BST

Author: Anthony Esmeralda, Kevin Ngo

Class: CSCI 220

Date: Novemember 14, 2017

Description: Binary Search Tree that uses an AVL tree search through records

of county/state, population, and county/state name

I certify that the code below is my own work.

Exception(s): N/A

\*/

#ifndef \_AVL\_TREE\_H\_

#define \_AVL\_TREE\_H\_

#include "SearchTree.h"

#include <cmath>

#include <algorithm>

// AVL Tree Class Definition

class AVLTree : public SearchTree

{

protected:

typedef int county\_state\_code; // a key

typedef BinaryTree::Position TPos; // a tree position

public: // public functions

AVLTree(); // constructor

Iterator insert(Entry& entry); //Insert the entry based on the key (county\_state\_code)

void erase(const county\_state\_code& key); // remove country\_state\_code's entry

void erase(const Iterator& it); // remove entry at it

protected: // utility functions

int height(const TPos& pos) const; // node height utility

void setHeight(TPos pos); // set height utility

bool isBalanced(const TPos& pos) const; // is the position balanced?

TPos tallGrandchild(const TPos& pos) const; // get tallest grandchild

void rebalance(const TPos& pos); // rebalance utility

};

#endif // !\_AVL\_TREE\_H\_

BinaryTree.h

/\* Program: Project 4 - BST

Author: Anthony Esmeralda, Kevin Ngo

Class: CSCI 220

Date: Novemember 14, 2017

Description: Binary Search Tree that uses an AVL tree search through records

of county/state, population, and county/state name

I certify that the code below is my own work.

Exception(s): N/A

\*/

#ifndef \_BINARY\_TREE\_H\_

#define \_BINARY\_TREE\_H\_

#include <iostream>

#include <list>

#include <iterator>

#include "Entry.h"

using namespace std;

class BinaryTree {

protected:

struct Node {

Entry element;

int height;

Node \*parent;

Node \*left;

Node \*right;

Node()

{

parent = nullptr;

left = nullptr;

right = nullptr;

height = 0;

}

};

// POSITION CLASS (NESTED CLASS)

public:

class Position {

private:

Node \*curNode;

public:

Position()

{

curNode = nullptr;

}

Position(Node \*passedNode)

{

curNode = passedNode;

}

void addElement(Entry data)

{

curNode->element = data;

}

Entry& operator\*()

{

return curNode->element;

}

Position left() const

{

return Position(curNode->left);

}

Position right() const

{

return Position(curNode->right);

}

Position parent() const

{

return Position(curNode->parent);

}

void setParent(const Position &p)

{

curNode->parent = p.curNode;

}

void setRightChild(const Position &p)

{

curNode->right = p.curNode;

}

void setLeftChild(const Position &p)

{

curNode->left = p.curNode;

}

bool isRoot()

{

return curNode->parent == NULL;

}

bool isExternal() const

{

return ((curNode->left == NULL) && (curNode->right == NULL));

}

bool operator==(const Position &p)

{

return curNode == p.curNode;

}

bool operator!=(const Position &p)

{

return curNode != p.curNode;

}

int getHeight() const

{

return curNode->height;

}

void setHeight(int h)

{

if (h >= 0)

curNode->height = h;

}

friend class BinaryTree;

};

typedef list<Position> PositionList; // CREATE A LIST NAMED POSITIONLIST TO HOLD OUR POSITIONS.

/////////////////////////////////////////////////////////

// The Binary Tree's Public and Protected Functions //

// the list of prototypes for all the //

// functions that is required //

// while using the binary //

// tree //

/////////////////////////////////////////////////////////

public:

BinaryTree();

int size() const;

Position root() const;

PositionList positions() const;

void addRoot();

void setRoot(const Position & p);

void expandExternal(const Position& p);

Position removeAboveExternal(const Position &p);

protected:

void inorder(Node \*curNode, PositionList& pl) const;

private:

Node \*\_root;

int counter;

};

#endif // !\_BINARY\_TREE\_H\_

SearchTree.h

/\* Program: Project 4 - BST

Author: Anthony Esmeralda, Kevin Ngo

Class: CSCI 220

Date: Novemember 14, 2017

Description: Binary Search Tree that uses an AVL tree search through records

of county/state, population, and county/state name

I certify that the code below is my own work.

Exception(s): N/A

\*/

#ifndef \_SEARCH\_TREE\_H\_

#define \_SEARCH\_TREE\_H\_

#include "BinaryTree.h"

class SearchTree {

private:

BinaryTree T;

int tSize;

///////////////////////////////////////

// Functions that we will be able //

// to use outside of //

// SearchTree //

// Class //

//////////////////////////////////////

public:

class Iterator;

SearchTree();

int size() const;

bool empty() const;

void erase(int key);

void erase(const Iterator &p);

int findDepth(int key);

Iterator find(int key);

Iterator insert(Entry data);

Iterator begin();

Iterator end();

protected:

BinaryTree::Position root() const;

BinaryTree::Position finder(int key,BinaryTree::Position &data);

int depth(BinaryTree::Position &data);

BinaryTree::Position inserter(Entry data);

BinaryTree::Position eraser(BinaryTree::Position data);

BinaryTree::Position restructure(BinaryTree::Position x);

void newRoot(BinaryTree::Position x);

///////////////////////////////////

// Iterator SubClass //

/////////////////////////////////

public:

class Iterator {

private:

BinaryTree::Position data;

public:

Iterator(const BinaryTree::Position input)

{

data = input;

}

const Entry operator\*()

{

return \*data;

}

bool operator==(const Iterator&p)

{

return data == p.data;

}

bool operator!=(const Iterator&p)

{

return data != p.data;

}

Iterator& operator++() // use inorder.

{

BinaryTree::Position w = data.right();

if (!w.isExternal())

{

do

{

data = w;

w = w.left();

} while (!w.isExternal());

}

else

{

w = data.parent();

while (data == w.right())

{

data = w;

w = w.parent();

}

data = w;

}

return \*this;

}

friend class SearchTree;

};

};

#endif // !\_SEARCH\_TREE\_H

Entry.h

/\* Program: Project 4 - BST

Author: Anthony Esmeralda, Kevin Ngo

Class: CSCI 220

Date: Novemember 14, 2017

Description: Binary Search Tree that uses an AVL tree search through records

of county/state, population, and county/state name

I certify that the code below is my own work.

Exception(s): N/A

\*/

#ifndef \_ENTRY\_H\_

#define \_ENTRY\_H\_

#include <iostream>

#include <string>

#include <iomanip>

using namespace std;

class Entry {

public:

int county\_state\_code;

string county\_state\_name;

int population;

public:

Entry();

Entry(int code, int pop, string name);

int getCode();

int getPop();

string getName();

void setName(string name);

void setCode(int code);

void setPop(int pop);

void printData();

};

#endif // !\_ENTRY\_H\_

BeginProgram.h

#ifndef \_BEGIN\_PROGRAM\_H

#define \_BEGIN\_PROGRAM\_H

#include "AVLTree.h"

#include <fstream>

#include <iostream>

using namespace std;

class BeginProgram {

private:

AVLTree oak;

SearchTree mahogany;

ofstream myFile;

public:

BeginProgram();

void start();

protected:

void fillTree(SearchTree st, AVLTree at, string fileName);

void stringToEntry(string s, Entry & e);

int menu();

void performAction(AVLTree& tree, int \_case);

void performAction(SearchTree& tree, int \_case);

};

#endif // !\_BEGIN\_PROGRAM\_H

Implementations:

AVLTree.cpp

/\* Program: Project 4 - BST

Author: Anthony Esmeralda, Kevin Ngo

Class: CSCI 220

Date: Novemember 14, 2017

Description: Binary Search Tree that uses an AVL tree search through records

of county/state, population, and county/state name

I certify that the code below is my own work.

Exception(s): N/A

\*/

#include "AVLTree.h"

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

// Starting by defining the utilities //

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

int AVLTree::height(const TPos& pos) const

{

if (pos.isExternal()) // If the node is equal to null, then return 0 because you are at the end.

return 0;

else

return pos.getHeight(); // Else will return the node's current height

}

// Will simply set h equal to the highest height of the left or right node

void AVLTree::setHeight(TPos pos)

{

int heightL = height(pos.left()); // Get the left node's height

int heightR = height(pos.right()); // Get the right node's height

pos.setHeight(1 + max(heightL,heightR)); // set the position as the max of the two

}

// Returns true if the position's height is balanced

bool AVLTree::isBalanced(const TPos& pos) const

{

TPos left = pos.left();

TPos right = pos.right();

int bal = height(left) - height(right); // Checks if the balance is over 1

return ((bal >= -1) && (bal <= 1)); // If it is over 1 or less than 1

}

// Returns the tallest grandchild

AVLTree::TPos AVLTree::tallGrandchild(const TPos& pos) const

{

TPos posL = pos.left();

TPos posR = pos.right();

if (height(posL) >= height(posR)) // If the height of the left position is greater than the height of the right's

{

if (height(posL.left()) >= height(posL.right())) // Check the height of the left position's children, if the left position's left child is greater than its right return the left that position

return posL.left();

else

return posL.right(); // Since the right child's height is greater, return the right

}

else // The height of the right position is greater than the height of the left's

{

if (height(posR.right()) >= height(posR.left())) // If the right position's height of the right child is greater than the left child return the right

return posR.right();

else // Return the left one since its greater

return posR.left();

}

}

// Rebalances the tree

void AVLTree::rebalance(const TPos& pos) // Will rebalance whatever position passed

{

TPos temp = pos; // Assigns a temporary position to the passed position

TPos whatRoot = root();

while (temp != root()) // While temp is not the root

{

whatRoot = root();

temp = temp.parent(); // Assign temp to be its parent

setHeight(temp); // set the height of the parent

if (!isBalanced(temp)) // If the node is unbalanced will balance it

{

TPos otherTemp = tallGrandchild(temp); // Sets another position as the tallest grandchild

temp = restructure(otherTemp); // Restructures that grandchild then assigns the rebalanced section to temp

setHeight(temp.left()); // Corrects the height

setHeight(temp.right());

setHeight(temp); // Sets temp's height

}

}

}

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

Starting by defining the public

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

// Will call the constructor of SearchTree since, there is no new data type (compared to the BST), but rather just functions in the AVLTree class

AVLTree::AVLTree() : SearchTree() {};

// Will insert an entry, then return an iterator at that position

AVLTree::Iterator AVLTree::insert(Entry& entry)

{

TPos temp = inserter(entry); // Inserts the entry then returns that position

setHeight(temp); // Sets the height at that position so it can be used before rebalancing

rebalance(temp);

return Iterator(temp);

}

// Erases a position in the AVLTree (parameter is a key which should be a county\_state\_code

void AVLTree::erase(const county\_state\_code& key)// Erase a key

{

TPos temp = finder(key, root());

if (!temp.isExternal()) // If the item is a external

{

TPos otherTemp = eraser(temp); // Erases the temp and returns the position of the position for rebalance

rebalance(otherTemp);

cout << "Record was succesfully deleted" << endl;

}

else cout << "Record doesnt exists" << endl;

}

BinaryTree.cpp

/\* Program: Project 4 - BST

Author: Anthony Esmeralda, Kevin Ngo

Class: CSCI 220

Date: Novemember 14, 2017

Description: Binary Search Tree that uses an AVL tree search through records

of county/state, population, and county/state name

I certify that the code below is my own work.

Exception(s): N/A

\*/

#include "BinaryTree.h"

BinaryTree::BinaryTree()

{

\_root = nullptr;

counter = 0;

}

int BinaryTree::size() const

{

return counter;

}

BinaryTree::Position BinaryTree::root() const

{

return Position(\_root);

}

BinaryTree::PositionList BinaryTree::positions() const

{

PositionList pl;

inorder(\_root, pl);

return PositionList(pl);

}

void BinaryTree::addRoot()

{

\_root = new Node;

counter++;

}

void BinaryTree::setRoot(const Position & p)

{

Node \*temp = p.curNode;

\_root = temp;

}

BinaryTree::Position BinaryTree::removeAboveExternal(const Position & p)

{

Node \*temp = p.curNode;

Node \*par = temp->parent;

Node \*sibling;

if (temp == par->left)

{

sibling = par->right;

}

else sibling = par->left;

if (par == \_root)

{

\_root = sibling;

sibling->parent = NULL;

}

else

{

Node \*grandparent = par->parent;

if (par == grandparent->left)

{

grandparent->left = sibling;

}

else grandparent->right = sibling;

sibling->parent = grandparent;

}

delete temp;

delete par;

counter -= 2;

return Position(sibling);

}

void BinaryTree::expandExternal(const Position &p)

{

Node \*curNode = p.curNode;

curNode->left = new Node;

curNode->right = new Node;

curNode->left->parent = curNode;

curNode->right->parent = curNode;

}

void BinaryTree::inorder(Node \*curNode, PositionList &pl) const

{

if (curNode->left != nullptr)

{

inorder(curNode->left, pl);

}

pl.push\_back(Position(curNode));

if (curNode->right != nullptr)

{

inorder(curNode->right, pl);

}

}

SearchTree.cpp

/\* Program: Project 4 - BST

Author: Anthony Esmeralda, Kevin Ngo

Class: CSCI 220

Date: Novemember 14, 2017

Description: Binary Search Tree that uses an AVL tree search through records

of county/state, population, and county/state name

I certify that the code below is my own work.

Exception(s): N/A

\*/

#include "SearchTree.h"

SearchTree::SearchTree()

{

T.addRoot();

tSize = 0;

T.expandExternal(T.root());

}

int SearchTree::size() const

{

return tSize;

}

bool SearchTree::empty() const

{

return tSize == 0;

}

void SearchTree::erase(int key)

{

BinaryTree::Position v = finder(key, root());

if (v.isExternal())

{

cout << "data was not found on " << key;

}

else eraser(v);

}

void SearchTree::erase(const Iterator & p)

{

eraser(p.data);

}

int SearchTree::findDepth(int key)

{

BinaryTree::Position value = finder(key, root());

int n = depth(value);

return n;

}

SearchTree::Iterator SearchTree::find(int key)

{

BinaryTree::Position value = finder(key, root()); // use the finder function to find our position

if (!value.isExternal()) // if our value function is internal

{

return SearchTree::Iterator(value); // return the iterator

}

else return SearchTree::Iterator(root().parent()); // else return NULL.

}

SearchTree::Iterator SearchTree::insert(Entry data)

{

BinaryTree::Position value = inserter(data); // use our inserter function to input our data

return SearchTree::Iterator(value); // return our iterator.

}

SearchTree::Iterator SearchTree::begin()

{

BinaryTree::Position v = root();

while (!v.isExternal())

{

v = v.left();

}

return Iterator(v.parent());

}

SearchTree::Iterator SearchTree::end()

{

return Iterator(T.root());

}

BinaryTree::Position SearchTree::root() const

{

return T.root().left(); // since T.root() is our superroot, we go to the left to get our root.

}

BinaryTree::Position SearchTree::finder(int key,BinaryTree::Position &data)

{

Entry dataEntry = \*data; // grab the data's position entry data.

if (data.isExternal()) // check if the data's position is external

{

return data; // then it was not found and return data position

}

if (key < dataEntry.county\_state\_code) // if the key is less than the data position key

{

return finder(key, data.left()); // recursively go to the left until the position key == key

}

else if (dataEntry.county\_state\_code < key) // if the key is greater than the position key

{

return finder(key, data.right()); // then recursively go to the right until the position key == key

}

else return data;

}

int SearchTree::depth(BinaryTree::Position & data)

{

int n = 1;

while (data != root())

{

data = data.parent();

n++;

}

return n;

}

BinaryTree::Position SearchTree::inserter(Entry data)

{

int key = data.county\_state\_code; // obtain the key from our Entry variable data

BinaryTree::Position value = finder(key, root());

while (!value.isExternal()) // while value is internal

{

value = finder(key, value.right()); // find a position to place our Entry variable data

}

Entry temp = \*value;

T.expandExternal(value); // expand that position

value.addElement(data); // and add the data into that position

tSize++; // increase our size

return value;

}

BinaryTree::Position SearchTree::eraser(BinaryTree::Position data)

{

BinaryTree::Position w; // create a position holder (we named it w)

if (data.left().isExternal()) // check if our data position left is external

{

w = data.left(); // if it is, then w = data.left

}

else if (data.right().isExternal()) // check if our data position right is external if left is not ex

{

w = data.right(); // if it is, then w = data.right

}

else // if none of the above

{

w = data.right(); // set w to data.right

do

{

w = w.left(); // keep looping left until we are at an external root (smallest right)

} while (!w.isExternal());

BinaryTree::Position u = w.parent(); // set u to be w's parent

data.addElement(\*u); // set the position u's data to be added into data's position's data to save

}

tSize--;

return T.removeAboveExternal(w);

}

BinaryTree::Position SearchTree::restructure(BinaryTree::Position x)

{ // data is our x variable

BinaryTree::Position y = x.parent(); // parent is our y variable

BinaryTree::Position z = y.parent(); // grandparent is our z variable

BinaryTree::Position a, b, c, t0, t1, t2, t3, newNode;

if (y == z.right() && x == y.right()) // if our tree is a single rotation case on the right side

{

a = z; // set our a,b,c accordingly

b = y;

c = x;

t0 = a.left(); // t0 is always a's left

t1 = b.left(); // t1 is always b's left

t2 = c.left(); // t2 is always c's left

t3 = c.right(); // t3 is always c's right

newNode = b; // set b to be our new subtree root

newNode.setParent(z.parent()); // set our newNode parents to be z's parent

if (z != z.parent().left()) // if z does not equal the parent of z's left

{

z.parent().setRightChild(newNode); // then newNode is the z's parent right child (since we are on the right side)

}

else z.parent().setLeftChild(newNode); // else then we are the left child since z was a root and the superoot's child is on the left side

t1.setParent(a); // set t1 parents to be a

a.setRightChild(t1); // and a's right child to be t1

a.setParent(newNode); // a's parent is our new Node

c.setParent(newNode); // c's parents is our new Node

newNode.setLeftChild(a); // set new Nodes left child to be a

newNode.setRightChild(c);

}

if (y == z.left() && x == y.left()) // if our a tree is a single rotation case on left side

{

a = z; // set our a,b,c accordingly

b = y;

c = x;

t0 = a.right(); // t0 is always a's right

t1 = b.right(); // t1 is always b's right

t2 = c.right(); // t2 is always c's right

t3 = c.left(); // t3 is always c's left

newNode = b; // b will be the new subtree root

newNode.setParent(z.parent()); // newNodes parent is z's parent

if (z.parent().left() == z)

{

z.parent().setLeftChild(newNode); // this one doesnt need a special case like in the right side case since we're always making z's parent child left.

}

else z.parent().setRightChild(newNode);

t1.setParent(a); // t1's parent is a

a.setLeftChild(t1); // a left child is t1

a.setParent(newNode); // a's parent is newNode

c.setParent(newNode); // c's parent is newNode

newNode.setRightChild(a); // newNode's right child is a

newNode.setLeftChild(c);

// again, we do not need to set a left child since newNode's left child is already c.

}

if (y == z.right() && x == y.left())

{

a = z; // set our a,b,c accordingly

b = x;

c = y;

t0 = a.left(); // t0 is always a's left

t1 = b.left(); // t1 is always b's right

t2 = b.right(); // t2 is always b's left

t3 = c.right(); // t3 is always c's right

newNode = b; // b will be the new subtree root

newNode.setParent(z.parent()); // newNodes parent is z's parents

if (z != z.parent().left()) // check if z is a root since we're doing a right left rotation

{

z.parent().setRightChild(newNode); // if its not, newNode will be the right child of z's parent

}

else z.parent().setLeftChild(newNode); // else it will be the new root

t1.setParent(a); // t1's parent is a

a.setRightChild(t1); // a's right child is t1

t2.setParent(c); // t2's parent is c

c.setLeftChild(t2); // c's left child is t2

a.setParent(newNode); // a's parent is newNode

c.setParent(newNode); // c's parent is newNode.

newNode.setLeftChild(a); // newNodes left child is a

newNode.setRightChild(c); // newNodes right child is b

}

if (y == z.left() && x == y.right())

{

c = z; // set our a,b,c accordingly

b = x;

a = y;

t0 = a.left(); // t0 is always a's left

t1 = b.left(); // t1 is always b's left

t2 = b.right(); // t2 is always b's right

t3 = c.right(); // t3 is always c's right

newNode = b; // b will be the new subtree root

newNode.setParent(z.parent()); // newNodes parent is z's parents

if (z.parent().left() == z)

{

z.parent().setLeftChild(newNode); // z's parent left child is our newNode

}

else z.parent().setRightChild(newNode);

t1.setParent(a); // t1's parent is a

a.setRightChild(t1); // a's right child is t1

t2.setParent(c); // t2's parent is c;

c.setLeftChild(t2); // c's left child is t2

a.setParent(newNode); // a's parent is newNode

c.setParent(newNode); // c's parent is newNode

newNode.setLeftChild(a); // newNodes left child is a

newNode.setRightChild(c); // newNodes right child is c

}

if (z == root()) // if z was the root, we need to make sure to change the root

{

newRoot(newNode);

}

return newNode;

}

void SearchTree::newRoot(BinaryTree::Position x)

{

T.setRoot(x);

}

Entry.cpp

/\* Program: Project 4 - BST

Author: Anthony Esmeralda, Kevin Ngo

Class: CSCI 220

Date: Novemember 14, 2017

Description: Binary Search Tree that uses an AVL tree search through records

of county/state, population, and county/state name

I certify that the code below is my own work.

Exception(s): N/A

\*/

#include "Entry.h"

Entry::Entry()

{

county\_state\_name = "";

county\_state\_code = 0;

population = 0;

}

Entry::Entry(int code, int pop, string name)

{

county\_state\_name = name;

county\_state\_code = code;

population = pop;

}

int Entry::getCode()

{

return county\_state\_code;

}

int Entry::getPop()

{

return population;

}

string Entry::getName()

{

return county\_state\_name;

}

void Entry::setName(string name)

{

county\_state\_name = name;

}

void Entry::setCode(int code)

{

county\_state\_code = code;

}

void Entry::setPop(int pop)

{

population = pop;

}

void Entry::printData()

{

setfill(" ");

cout << left << setw(20) << county\_state\_code << setw(20) << population << setw(15) << left << county\_state\_name << right;

}

BeginProgram.cpp

#include "BeginProgram.h"

BeginProgram::BeginProgram()

{

fillTree(mahogany, oak, "p4Large.txt");

}

void BeginProgram::start()

{

Entry ent;

cout << "success!\n\n";

int option;

char optionTwo;

do

{

cout << "Select A to work with Search Tree and B to work with AVL Tree (AVLTree will be selected if did not choose A or B): ";

cin >> optionTwo;

if (toupper(optionTwo) == 'A')

{

cout << "You are in the BST Tree\n";

option = menu();

if ((option != 5) && (option != 0)) // If option is 0, will just rerun the loop, only 5 will terminate this

{

performAction(mahogany, option);

}

}

else

{

cout << "You are in the AVL Tree\n";

option = menu();

if ((option != 5) && (option != 0)) // If option is 0, will just rerun the loop, only 5 will terminate this

{

performAction(oak, option);

}

}

} while (option != 5);

}

void BeginProgram::fillTree(SearchTree st, AVLTree at, string fileName)

{

Entry E;

string record;

fstream infile;

infile.open(fileName);

if (infile.is\_open())

{

while (!infile.eof())

{

getline(infile, record);

stringToEntry(record, E);

st.insert(E);

at.insert(E);

}

}

else

cout << "Incorrect file name passed or does not exist.\n";

infile.close();

}

void BeginProgram::stringToEntry(string s, Entry &e)

{

int sSize = s.length(), lowerBound = 0, upperBound, dummyV;

bool commaOne = false, commaTwo = false;

string subString;

for (int i = 0; i < sSize; i++)

{

if ((s[i] == ',') && (commaOne != true))

{

commaOne = true;

upperBound = i;

subString = s.substr(lowerBound, upperBound - lowerBound);

lowerBound = ++upperBound;

dummyV = atoi(subString.c\_str());

e.county\_state\_code = dummyV;

}

else if ((s[i] == ',') && (commaTwo != true) && (commaOne == true))

{

commaTwo = true;

upperBound = i;

subString = s.substr(lowerBound, upperBound - lowerBound);

lowerBound = upperBound + 2;

dummyV = atoi(subString.c\_str());

e.population = dummyV;

}

if ((s[i] == '\"') && (commaOne && commaTwo))

{

upperBound = i;

subString = s.substr(lowerBound, upperBound - lowerBound);

e.county\_state\_name = subString;

}

}

}

int BeginProgram::menu()

{

string input;

char inputAsChar;

cout << "1. Search for a record\n2. Insert a record\n3. Delete a record\n4. List all records\n5. Exit\n";

cout << "input: ";

cin >> input;

inputAsChar = input[0];

switch (inputAsChar)

{

case '1':

return 1;

break;

case '2':

return 2;

break;

case '3':

return 3;

break;

case '4':

return 4;

case '5':

return 5;

default:

return 0;

}

}

void BeginProgram::performAction(AVLTree& tree, int \_case)

{

int countySC, population;

string name;

if (\_case == 1)

{

cout << "You chose to search for a record, enter a county-state-code: ";

cin >> countySC;

SearchTree::Iterator found = tree.find(countySC);

Entry element = \*found;

cout << endl;

if (element.county\_state\_code != 0)

{

cout << left << setw(20) << "county state code" << setw(20) << "population" << setw(20) << left << "county state name";

cout << endl;

cout << "----------------------------------------------------------------------------\n";

element.printData();

cout << endl << endl;

int runTime = tree.findDepth(countySC);

cout << "RunTime: " << runTime << " milli-seconds" << endl;

}

else cout << "No data found" << endl;

cout << endl;

}

if (\_case == 2)

{

cout << "You chose to insert a record\n";

cout << "Enter county-state-code: ";

cin >> countySC;

cout << "Enter population: ";

cin >> population;

cout << "Enter the state/county name: ";

cin.ignore();

getline(cin, name);

Entry element(countySC, population, name);

tree.insert(element);

int runTime = tree.findDepth(countySC);

cout << "RunTime: " << runTime << " milli-seconds" << endl;

cout << "Succesfully entered your record\n";

}

if (\_case == 3)

{

cout << "You chose to delete a record\n";

cout << "Enter which record you would like to delete by county-state-code: ";

cin >> countySC;

int runTime = tree.findDepth(countySC);

cout << endl;

cout << "RunTime: " << runTime << " milli - seconds" << endl << endl;

tree.erase(countySC);

}

if (\_case == 4)

{

SearchTree::Iterator it(tree.begin());

Entry output;

setfill(" ");

myFile.open("AVLoutput.txt");

cout << left << setw(20) << "county state code" << setw(20) << "population" << setw(20) << left << "county state name" << endl;

cout << "----------------------------------------------------------------------------\n";

myFile << left << setw(20) << "county state code" << setw(20) << "population" << setw(20) << left << "county state name" << endl;

myFile << "----------------------------------------------------------------------------\n";

for (it; it != tree.end(); ++it)

{

output = \*it;

countySC = output.getCode();

population = output.getPop();

name = output.getName();

cout << left << setw(20) << countySC << setw(20) << population << setw(15) << left << name << right << endl;

myFile << left << setw(20) << countySC << setw(20) << population << setw(15) << left << name << right << endl;

}

myFile.close();

cout << endl;

}

}

void BeginProgram::performAction(SearchTree& tree, int \_case)

{

int countySC, population;

string name;

if (\_case == 1)

{

cout << "You chose to search for a record, enter a county-state-code: ";

cin >> countySC;

SearchTree::Iterator found = tree.find(countySC);

Entry element = \*found;

cout << endl;

if (element.county\_state\_code != 0)

{

cout << left << setw(20) << "county state code" << setw(20) << "population" << setw(20) << left << "county state name";

cout << endl;

cout << "----------------------------------------------------------------------------\n";

element.printData();

cout << endl << endl;

int runTime = tree.findDepth(countySC);

cout << "RunTime: " << runTime << " milli - seconds" << endl;

}

else cout << "No data found" << endl;

cout << endl;

}

if (\_case == 2)

{

cout << "You chose to insert a record\n";

cout << "Enter county-state-code: ";

cin >> countySC;

cout << "Enter population: ";

cin >> population;

cout << "Enter the state/county name: ";

cin.ignore();

getline(cin, name);

Entry element(countySC, population, name);

tree.insert(element);

int runTime = tree.findDepth(countySC);

cout << "RunTime: " << runTime << " milli - seconds" << endl;

cout << "Succesfully entered your record\n";

}

if (\_case == 3)

{

cout << "You chose to delete a record\n";

cout << "Enter which record you would like to delete by county-state-code: ";

cin >> countySC;

int runTime = tree.findDepth(countySC);

cout << endl;

cout << "RunTime: " << runTime << " milli - seconds" << endl << endl;

tree.erase(countySC);

}

if (\_case == 4)

{

SearchTree::Iterator it(tree.begin());

Entry output;

setfill(" ");

cout << endl;

myFile.open("BSToutput.txt");

cout << left << setw(20) << "county state code" << setw(20) << "population" << setw(20) << left << "county state name" << endl;

cout << "----------------------------------------------------------------------------\n";

myFile << left << setw(20) << "county state code" << setw(20) << "population" << setw(20) << left << "county state name" << endl;

myFile << "----------------------------------------------------------------------------\n";

for (it; it != tree.end(); ++it)

{

output = \*it;

countySC = output.getCode();

population = output.getPop();

name = output.getName();

cout << left << setw(20) << countySC << setw(20) << population << setw(15) << left << name << right << endl;

myFile << left << setw(20) << countySC << setw(20) << population << setw(15) << left << name << right << endl;

}

myFile.close();

cout << endl;

}

}

Driver(s):

Main.cpp

/\* Program: Project 4 - BST

Author: Anthony Esmeralda, Kevin Ngo

Class: CSCI 220

Date: Novemember 14, 2017

Description: Binary Search Tree that uses an AVL tree search through records

of county/state, population, and county/state name

I certify that the code below is my own work.

Exception(s): N/A

\*/

#include <iostream>

#include <fstream>

#include <string>

#include "BeginProgram.h"

using namespace std;

int main()

{

cout << "Authors: Kevin Ngo & Anthony Esmeralda\n";

cout << "Planting the AVL/BS tree(s).....";

BeginProgram begin;

begin.start();

return 0;

}