**Sabastian Fasano**

**February 9, 2025**

**CS-300**

**Module 5 BinarySearchTree**

The code is divided into the following functions/structs/classes:

• **Class BinarySearchTree**

* Class definition for BinarySearchTree objects that contain:
  + **Private members**: size, root, and inOrder()
  + **Public methods**: Constructor, Destructor, InOrder(), Insert(), Remove(), Search(), Size()

• **BinarySearchTree()**

* A default constructor that initializes the root as a null pointer.

• **BinarySearchTree::Insert(Bid)**

* Accepts a Bid as input.
* The function then adds the Bid as a new Node within the tree (nodes).

• **BinarySearchTree::InOrder()**

* This function traverses the BinarySearchTree starting at the root and outputs four values from the Bid struct to the console.

• **BinarySearchTree::Remove(String)**

* Begins at the root and searches for the provided String.
* Once the String is found, the Node holding it is deallocated from memory.
* The tree is updated accordingly.

• **BinarySearchTree::Search(String)**

* Starts at the root and searches for the given String.
* If the String is found, the Node containing it is returned.

• **BinarySearchTree::Size()**

* A getter method used to retrieve the private member, size.

• **strToDouble**

* Converts data from the CSV file into a usable numeric value.

• **Bid**

* A struct that contains the data.
* Used with a vector that will later be sorted.

• **Node**

* A struct that holds the Node data.
* It consists of a Bid and two pointers: left and right.

• **loadBids**

* This function is responsible for reading CSV data.
* It can read the CSV path from arguments or fall back to a default path.
* The data is then parsed into the Bid structure, and each Bid is added to the BinarySearchTree.

• **main**

* The main function serves as the primary driver for the application.
* It presents a menu that allows the user to input a bid, load data, view data, delete a node, and then exit the program.
* It also reports the time taken for each algorithm using the time.h library.

**end**

The code was relatively simple to understand, especially since the parser was provided.

**Pseudocode:**

Main Function()

Read command line arguments

Store the argument as the CSV file path

If no command line arguments, load the default CSV file path

Loop while choice is not equal to ‘9’

Output menu

Get user input and store it in choice

Validate user input

If choice is not between 1-4 or 9, throw an error

If choice equals ‘1’

Start the clock and store time in ticks

Call loadBids and store CSV data in BinarySearchTree bst

Output the number of records in the CSV file

Stop the clock

Output the elapsed time taken to load the CSV file

If choice equals ‘2’

Call inOrder()

If choice equals ‘3’

Start the clock and store time in ticks

Call Search() passing a bidKey to search for

Stop the clock

Output the elapsed time taken to find the bidKey

If choice equals ‘4’

Call Remove() passing bidKey

If choice equals ‘9’

Exit the application

Output ‘Goodbye’

End

BinarySearchTree::Insert(Bid)

Create a new Node pointer called currentNode and set it to the root

Check if the root is Null

If the root is Null, create a new Node with the Bid

If the root is not Null

Loop until currentNode is Null

If the bidId within Bid is less than the currentNode’s bidId

If the currentNode’s left pointer is Null

Set the currentNode’s left pointer to a new Node containing the Bid

Set currentNode to Null to break

Else, set currentNode to the Node at the left pointer

Else

If the currentNode’s right pointer is Null

Set the currentNode’s right pointer to a new Node containing the Bid

Set currentNode to Null to break

Else, set currentNode to the Node at the right pointer

Increment size

End

BinarySearchTree::InOrder(Node)

If Node is Null, return

Recursively call Node’s left pointer to reach the leftmost Node

Output the bidId, title, amount, and fund to the console

Recursively call Node’s right pointer to find the rightmost Node

End

BinarySearchTree::Search(String)

Create a new Node pointer called currentNode

Set currentNode to the root

Loop until currentNode is Null (end of the tree)

If currentNode contains a bidId equal to the String

Return the Bid at currentNode

If currentNode’s bidId is less than the String

Set currentNode to the left Node

Else, set currentNode to the right Node

Return an empty Bid

End

BinarySearchTree::Remove(String)

Create a new Node pointer called curr and set it to the root

Create a new Node pointer called par and set it to Null

Loop until curr is Null

If curr contains a BidId equal to the String

Check if there are leaves

If there are leaves, assign them to Par

Create a new Node called suc and set it to curr’s right pointer

Loop until suc’s left pointer is Null

Set suc to its left pointer

Create a new Node called successorData

Set successorData to a node containing suc

Recursively call Remove() passing suc’s bidId

Set curr’s bid equal to successorData’s bid

If curr’s bidId is less than the String

Set par to curr

Set curr to curr’s right pointer

Else

Set par to curr

Set curr to curr’s left pointer

End

**Code:**

//============================================================================

// Name : BinarySearchTree.cpp

// Author : Sabastian Fasano

// Version : 1.0

// Copyright : Copyright © 2022 SNHU COCE

// Description : Hello World in C++, Ansi-style

//============================================================================

#include <iostream>

#include <time.h>

#include <Windows.h>

#include "CSVparser.hpp"

using namespace std;

//============================================================================

// Global definitions visible to all methods and classes

//============================================================================

// forward declarations

double strToDouble(string str, char ch);

// define a structure to hold bid information

struct Bid {

string bidId; // unique identifier

string title;

string fund;

double amount;

Bid() {

amount = 0.0;

}

};

// Internal structure for tree node

struct Node {

Bid bid;

Node\* left;

Node\* right;

// default constructor

Node() {

left = nullptr;

right = nullptr;

}

// initialize with a bid

Node(Bid aBid) {

bid = aBid;

left = nullptr;

right = nullptr;

}

//// create the key for the given bid

//unsigned int key = hash(atoi(bidId.c\_str()));

//Node\* node = &(nodes.at(key));

//// if entry found for the key

//if (node != nullptr && node->key != UINT\_MAX

// && node->bid.bidId.compare(bidId) == 0) {

// return node->bid;

//}

//// if no entry found for the key

//if (node == nullptr || node->key == UINT\_MAX) {

// return bid;

//}

//while (node != nullptr) {

// // if the current node matches, return it

// if (node->key != UINT\_MAX && node->bid.bidId.compare(bidId) == 0) {

// return node->bid;

// }

// node = node->next;

//}

//return bid;

};

//============================================================================

// Binary Search Tree class definition

//============================================================================

/\*\*

\* Define a class containing data members and methods to

\* implement a binary search tree

\*/

class BinarySearchTree {

private:

Node\* root;

//void addNode(Node\* node, Bid bid);

void inOrder(Node\* node);

//Node\* removeNode(Node\* node, string bidId);

int size = 0;

public:

BinarySearchTree();

virtual ~BinarySearchTree();

void InOrder();

void Insert(Bid bid);

void Remove(string bidId);

Bid Search(string bidId);

int Size();

};

/\*\*

\* Default constructor

\*/

BinarySearchTree::BinarySearchTree() {

// FixMe (1): initialize housekeeping variables

//root is equal to nullptr

this->root = nullptr;

}

/\*\*

\* Destructor

\*/

BinarySearchTree::~BinarySearchTree() {

// recurse from root deleting every node

}

/\*\*

\* Traverse the tree in order

\*/

void BinarySearchTree::InOrder() {

// FixMe (2): In order root

// call inOrder fuction and pass root

inOrder(root);

}

/\*\*

\* Traverse the tree in post-order

\*/

//void BinarySearchTree::PostOrder() {

// FixMe (3): Post order root

// postOrder root

//}

/\*\*

\* Traverse the tree in pre-order

\*/

//void BinarySearchTree::PreOrder() {

// // FixMe (4): Pre order root

// // preOrder root

//}

/\*\*

\* Insert a bid

\*/

void BinarySearchTree::Insert(Bid bid) {

// FIXME (5) Implement inserting a bid into the tree

// if root equarl to null ptr

// root is equal to new node bid

// else

// add Node root and bid

Node\* currentNode = root;

if (root == NULL) {

root = new Node(bid);

}

else {

while (currentNode != NULL) {

if (bid.bidId < currentNode->bid.bidId) {

if (currentNode->left == nullptr) {

currentNode->left = new Node(bid);

currentNode = NULL;

}

else {

currentNode = currentNode->left;

}

}

else {

if (currentNode->right == nullptr) {

currentNode->right = new Node(bid);

currentNode = NULL;

}

else {

currentNode = currentNode->right;

}

}

}

}

size++;

}

/\*\*

\* Remove a bid

\*/

void BinarySearchTree::Remove(string bidId) {

// FIXME (6) Implement removing a bid from the tree

// remove node root bidID

Node\* par = NULL;

Node\* curr = root;

while (curr != NULL) {

if (curr->bid.bidId == bidId) {

if (curr->left == NULL && curr->right == NULL) {

if (par == NULL) {

root = nullptr;

}

else if (par->left == curr) {

par->left = NULL;

}

else {

par->right = NULL;

}

}

else if (curr->right == NULL) {

if (par == NULL) {

root = curr->left;

}

else if (par->left == curr) {

par->left = curr->left;

}

else {

par->right = curr->left;

}

}

else if (curr->left == NULL) {

if (par == NULL) {

root = curr->right;

}

else if (par->left == curr) {

par->left = curr->right;

}

else {

par->right = curr->right;

}

}

else {

Node\* suc = curr->right;

while (suc->left != NULL) {

suc = suc->left;

}

Node successorData = Node(suc->bid);

Remove(suc->bid.bidId);

curr->bid = successorData.bid;

}

//cout << "\nNode found and removed" << endl;

return;

}

else if (curr->bid.bidId < bidId) {

par = curr;

curr = curr->right;

}

else {

par = curr;

curr = curr->left;

}

}

cout << "\nValue not found" << endl;

return;

}

/\*\*

\* Search for a bid

\*/

Bid BinarySearchTree::Search(string bidId) {

// FIXME (7) Implement searching the tree for a bid

// set current node equal to root

// keep looping downwards until bottom reached or matching bidId found

// if match found, return current bid

// if bid is smaller than current node then traverse left

// else larger so traverse right

Bid bid;

Node\* currentNode = root;

while (currentNode != NULL) {

if (currentNode->bid.bidId == bidId) {

return currentNode->bid;

}

else if (bidId < currentNode->bid.bidId) {

currentNode = currentNode->left;

}

else {

currentNode = currentNode->right;

}

}

//not found

//cout << "Value not found." << endl;

return bid;

}

/\*\*

\* Add a bid to some node (recursive)

\*

\* @param node Current node in tree

\* @param bid Bid to be added

\*/

void BinarySearchTree::addNode(Node\* node, Bid bid) {

// FIXME (8) Implement inserting a bid into the tree

// if node is larger then add to left

// if no left node

// this node becomes left

// else recurse down the left node

// else

// if no right node

// this node becomes right

//else

// recurse down the left node

}

void BinarySearchTree::inOrder(Node\* node) {

// FixMe (9): Pre order root

//if node is not equal to null ptr

//InOrder not left

//output bidID, title, amount, fund

//InOder right

if (node == NULL) {

return;

}

inOrder(node->left);

//print the node

cout << node->bid.bidId << ": " << node->bid.title << " | " << node->bid.amount

<< " | " << node->bid.fund << endl;

inOrder(node->right);

}

//void BinarySearchTree::postOrder(Node\* node) {

// // FixMe (10): Pre order root

// //if node is not equal to null ptr

// //postOrder left

// //postOrder right

// //output bidID, title, amount, fund

//

//}

//

//void BinarySearchTree::preOrder(Node\* node) {

// // FixMe (11): Pre order root

// //if node is not equal to null ptr

// //output bidID, title, amount, fund

// //postOrder left

// //postOrder right

//}

int BinarySearchTree::Size() {

return size;

}

//============================================================================

// Static methods used for testing

//============================================================================

/\*\*

\* Display the bid information to the console (std::out)

\*

\* @param bid struct containing the bid info

\*/

void displayBid(Bid bid) {

cout << bid.bidId << ": " << bid.title << " | " << bid.amount << " | "

<< bid.fund << endl;

return;

}

/\*\*

\* Load a CSV file containing bids into a container

\*

\* @param csvPath the path to the CSV file to load

\* @return a container holding all the bids read

\*/

void loadBids(string csvPath, BinarySearchTree\* bst) {

cout << "Loading CSV file " << csvPath << endl;

// initialize the CSV Parser using the given path

csv::Parser file = csv::Parser(csvPath);

// read and display header row - optional

vector<string> header = file.getHeader();

//for (auto const& c : header) {

// cout << c << " | ";

//}

//cout << "" << endl;

try {

// loop to read rows of a CSV file

for (unsigned int i = 0; i < file.rowCount(); i++) {

// Create a data structure and add to the collection of bids

Bid bid;

bid.bidId = file[i][1];

bid.title = file[i][0];

bid.fund = file[i][8];

bid.amount = strToDouble(file[i][4], '$');

//cout << "Item: " << bid.title << ", Fund: " << bid.fund << ", Amount: " << bid.amount << endl;

// push this bid to the end

bst->Insert(bid);

}

} catch (csv::Error &e) {

std::cerr << e.what() << std::endl;

}

}

/\*\*

\* Simple C function to convert a string to a double

\* after stripping out unwanted char

\*

\* credit: http://stackoverflow.com/a/24875936

\*

\* @param ch The character to strip out

\*/

double strToDouble(string str, char ch) {

str.erase(remove(str.begin(), str.end(), ch), str.end());

return atof(str.c\_str());

}

/\*\*

\* The one and only main() method

\*/

int main(int argc, char\* argv[]) {

// process command line arguments

string csvPath, bidKey;

switch (argc) {

case 2:

csvPath = argv[1];

bidKey = "98109";

break;

case 3:

csvPath = argv[1];

bidKey = argv[2];

break;

default:

csvPath = "eBid\_Monthly\_Sales.csv";

//bidKey = "98061";

bidKey = "97990"; //root

}

// Define a timer variable

clock\_t ticks;

// Define a binary search tree to hold all bids

BinarySearchTree\* bst = new BinarySearchTree();

Bid bid;

const int GLOBAL\_SLEEP\_VALUE = 5000;

int choice = 0;

string anyKey = " ";

bool goodInput;

while (choice != 9) {

cout << "Menu:" << endl;

cout << " 1. Load Bids" << endl;

cout << " 2. Display All Bids" << endl;

cout << " 3. Find Bid" << endl;

cout << " 4. Remove Bid" << endl;

cout << " 9. Exit" << endl;

cout << "Enter choice: ";

try {

cin >> choice;

if ((choice > 0 && choice < 5) || (choice == 9)) {// limit the user menu inputs to good values

goodInput = true;

}

else {//throw error for catch

goodInput = false;

throw 1;

}

switch (choice) {

case 1:

//bst = new BinarySearchTree();

// Initialize a timer variable before loading bids

ticks = clock();

// Complete the method call to load the bids

loadBids(csvPath, bst);

cout << bst->Size() << " bids read" << endl;

// Calculate elapsed time and display result

ticks = clock() - ticks; // current clock ticks minus starting clock ticks

cout << "time: " << ticks << " clock ticks" << endl;

cout << "time: " << ticks \* 1.0 / CLOCKS\_PER\_SEC << " seconds" << endl;

Sleep(GLOBAL\_SLEEP\_VALUE);

break;

case 2:

bst->InOrder();

cout << "\nEnter \'y\' to continue..." << endl;

cin >> anyKey;

break;

case 3:

ticks = clock();

bid = bst->Search(bidKey);

ticks = clock() - ticks; // current clock ticks minus starting clock ticks

if (!bid.bidId.empty()) {

displayBid(bid);

}

else {

cout << "Bid Id " << bidKey << " not found." << endl;

}

cout << "time: " << ticks << " clock ticks" << endl;

cout << "time: " << ticks \* 1.0 / CLOCKS\_PER\_SEC << " seconds" << endl;

Sleep(GLOBAL\_SLEEP\_VALUE);

break;

case 4:

bst->Remove(bidKey);

Sleep(GLOBAL\_SLEEP\_VALUE);

break;

case 9:

break;

default:

throw 2;

}

}

catch (int err) {

std::cout << "\nPlease check your input." << endl;

Sleep(GLOBAL\_SLEEP\_VALUE);

}

//need to clear the cin operator of extra input, e.g., 9 9, or any errors generated by bad input, e.g., 'a'

cin.clear();

cin.ignore();

//clear the consolse to redraw a fresh menu

system("cls");

}

cout << "Good bye." << endl;

Sleep(GLOBAL\_SLEEP\_VALUE);

return 0;

}