|  |  |  |
| --- | --- | --- |
| American University of Sharjah  College of Engineering  Department of Computer Science & Engineering  P. O. Box 26666, Sharjah, UAE |  | **Instructors:** Dr. Michel Pasquier  **Lab Instructor:** Praveena Kolli  **Office:** EB2-12  **Phone**: 971-6-5152352  **e-mail**: pkolli@aus.edu  **Semester**: Spring 2021 |

**CMP305L - Data Structures and Algorithms Lab**

**Lab. Assignment 11- BST**

***Objectives***

* To understand and practice coding Binary Search Trees
* To implement a simple real-world application of BST’s

***Instructions***

* Do not use any static or global variable. Use recursion!

***Note:***

***Lab:*** Exercises 1 and 2 (10 marks)

***Bonus*:** Exercise 3 (1 mark)

***Exercise 1:***

1. Write a member function that creates a *mirror* of a given Binary Search Tree of integers. A mirror tree is one where the structure is identical but reversed left-to-right. Moreover, for the mirror tree to be a valid BST, all values must be negated.

For example, if the BST shown on the left is given as input, the mirror function should return the BST on the right:

**BST**

**Mirror**

CODE:

void mirror(BinaryNode \*& t) {

if (t == nullptr) return;

mirror(t->left);

mirror(t->right);

//Negate the value:

t->element = -1 \* t->element;

//Swap left and right subtrees:

BinaryNode\* temp;

temp = t->left;

t->left = t->right;

t->right = temp;

}

Main.cpp:

#include<iostream>

#include"BinarySearchTree.h"

using namespace std;

int main() {

BinarySearchTree<int> bst{};

bst.insert(5);

bst.insert(2);

bst.insert(3);

bst.insert(1);

bst.insert(8);

bst.insert(6);

bst.insert(7);

cout << "BST Before Mirror: " << endl;

bst.printTree();

cout << "BST After Mirror: " << endl;

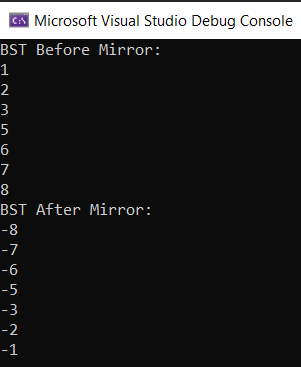
bst.mirror();

bst.printTree();

return 0;

}

SCREENSHOT:



1. Write a member function that returns true if a given Binary Search Tree is full, and false otherwise. The function should call a helper function that performs a single traversal of the BST to compute both its height H and size S. Then it checks whether S == 2H+1-1.

Helper function return a pair<int,int> with height and size as its values.

CODE:

bool isFull()

{

pair<int, int> x = calculateHeightAndSize(this->root );

return(x.second == (pow(2, x.first + 1) - 1));

}

pair<int, int> calculateHeightAndSize(BinaryNode\* t, int height = 0)

{

if (t != nullptr)

{

pair<int, int> x = calculateHeightAndSize(t->left, height + 1);

pair<int, int> y = calculateHeightAndSize(t->right, height + 1);

pair<int, int> ans;

ans.first = (x.first > y.first ? x.first : y.first);

ans.second = (x.second + y.second + 1);

return ans ;

}

else

{

pair<int, int> x2;

x2.first = height-1; x2.second = 0;

return x2;

}

}

Main.cpp:

#include<iostream>

#include"BinarySearchTree.h"

using namespace std;

int main() {

BinarySearchTree<int> bst{};

bst.insert(5);

bst.insert(2);

bst.insert(3);

bst.insert(1);

bst.insert(8);

bst.insert(6);

bst.insert(9);

cout << "BST Before Mirror: " << endl;

bst.printTree();

cout << "BST After Mirror: " << endl;

bst.mirror();

bst.printTree();

cout << endl << endl;

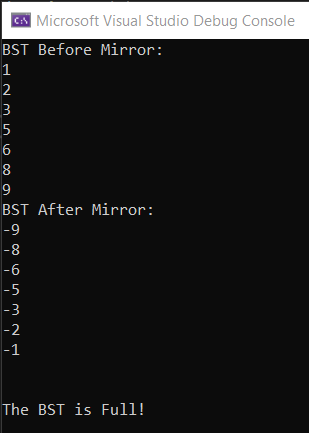
if (bst.isFull()) cout << "The BST is Full!" << endl;

else cout << "The BST is NOT Full!" << endl;

return 0;

}

SCREENSHOT:



**Note**: The BinarySearchTree.h file is provided to you.

***Exercise 2***

The provided *data.txt* file contains list of student information formatted as follows:

*Student ID (3 columns): first Name last Name (17 columns): GPA (1 column):*

Each student is represented with a *data record* that has *24* characters (inclusive of the character ‘:’) Students’ information is not sorted in any way.

Reading a file line by line searching for a particular *data record* is very slow, whereas using *seek()* is very fast. In this application we use a *Binary Search Tree* as an index to the file, to determine a record location in *O(log N)* time, then seek the *data record* in near constant time (instead of a slow *O(N)* retrieval).  The extra effort that is involved in creating a *Binary Search Tree* based on the *instances* of the class *RecordLocation* reallypays in time (speeds up the retrieval of records).

Create a RecordLocation class as follows:

class RecordLocation

{

private:

int studentId;//student id

int studentRecordNumber;// Student record number

public:

//provide constructors

//provide getters and setters

//provide overloaded opeartors <, > == based on student id.

//provide print function to print student id and student record

//number

};

Note that the “studentRecordNumber” for the *1st* student record in the file data.txt is *0.* The “studentRecordNumber” for the *11th* student record is *10*. To get the *11th* student’s details from the file *data.txt,* you must call seekg(24\*10) to skip 240 characters and read 24 characters from the 241st location.

Develop a program to create a *Binary Search Tree* such that each node has an *element* that is an instance of the class *RecordLocation*. Your program must ask the user to enter a student’s id, use this student’s id to get the student’s record number from the *Binary Serach Tree* that is already created. The record number must then be used to get student’s details from *data.txt* file and print the student’s details.

Use the following code as an example to understand how to advance to a specific record and read it.

#include <iostream>

#include <fstream>

using namespace std;

void main(){

char buffer[25] = {'0'};

ifstream input("data.txt", ios::binary);

// advance to the fourth record, then read

input.seekg(24\*3);

input.read (buffer,24);

cout << buffer << endl;

// advance (go back) to the first record

input.seekg(0);

input.read (buffer,24);

cout << buffer << endl;

input.close();

}

**CODE:**

**BinarySearchTree.h: (Changes only)**

Comparable contains2(const Comparable& x) const

{

return contains2(x, root);

}

Comparable contains2(const Comparable& x, BinaryNode\* t) const

{

if (t == nullptr)

return NULL;

else if (x < t->element)

return contains(x, t->left);

else if (t->element < x)

return contains(x, t->right);

else

return x; // Match

}

**Main.cpp:**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include<iostream>

#include"BinarySearchTree.h"

#include <fstream>

#include<string>

using namespace std;

class RecordLocation

{

private:

int studentId;//student id

int studentRecordNumber;// Student record number

public:

RecordLocation(int sID = 0, int srn = 0) {

studentId = sID; studentRecordNumber = srn;

}

void setStudentID(int s) { studentId = s; }

void setStudentRecordNumber(int s) { studentRecordNumber = s; }

int getStudentId() { return studentId; }

int getStudentRecordNumber() {return studentRecordNumber;}

bool operator<(const RecordLocation& rhs) const {

return (studentId < rhs.studentId);

}

bool operator>(const RecordLocation& rhs) const {

return (studentId > rhs.studentId);

}

bool operator==(const RecordLocation& rhs) const {

return (studentId == rhs.studentId);

}

friend ostream& operator<<(ostream& os, const RecordLocation&rhs) {

os << "Student ID: " << rhs.studentId << " Student Record Number: " << rhs.studentRecordNumber << endl;

return os;

}

};

int main() {

char buffer[25] = { '0' };

ifstream input("data.txt", ios::binary);

if (input.fail()) { cout << "Failed!"<<endl; }

BinarySearchTree<RecordLocation> bst{};

int i = 0;

while (!input.eof()) {

input.seekg(24 \* i);

input.read(buffer, 24);

char id[3];

strncpy(id, buffer, 3);

int myid = stoi(id);

RecordLocation temp(myid, i);

bst.insert(temp);

i++;

}

//bst.printTree();

cout << "Tree created!" << endl << endl;

cout << "Enter the userID to search for: " << endl;

int target;

cin >> target;

if (bst.contains(target)) {

RecordLocation ans = bst.contains2(target);

int index = ans.getStudentRecordNumber();

input.seekg(24 \* index);

input.read(buffer, 24);

cout << "The record is: " << endl << buffer << endl;

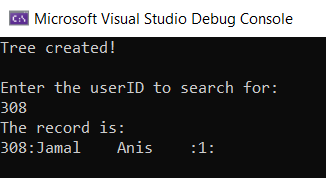
}

input.close();

return 0;

}

**SCREENSHOT:**



**Bonus:**

***Exercise 3:***

1. Write a member function, int childrenDiff(int threshold) that calculates the number of binary nodes whose children have a difference greater than the threshold.
2. Write a member function, bool isSumOfPath(int sum) that returns true if the tree has a root to leaf path such that adding up all the values along the path makes up the given sum.

A set of medical equipment

Description automatically generated with low confidence

For the above BST the expected output of the function callbst*.*isSumOfPath(22) is true. Sum of the elements of the nodes in the highlighted path from root 5 to leaf 2 is 22.