Project One

Ashlee WOod

Reading/PARSE/CLOSE the data from File:

Use fstream to open file:

Void intakeCourses(string cvsPath, StructureData)

Input call to open file up,

IF the return value is “-1”, then the file is not found

Else file is found

While loop till not the EOF

Parse each line

IF 0 > 2 values in a line,

return ERROR

ELSE read

IF third parameter,

IF 3 OR more parameter is in 1st parameter elsewhere,

continue

ELSE return Error

Close the file

Hold Course Info:

Make struct Course{}

Vector intakeCourses(string csvPath)

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

Intialize a data structure and add to the collection of courses

Course course;

course.courseId = file[i][1];

course.name = file[i][0];

while not end of the line;

course.prereq. = file[i][8];

courses.push\_back(course);

HASHTABLE:

//HashTable : create course objects Hashtable

Generate Hashtable

Generate Node struct

Course course

Unsigned int key

Vector< nodes>

Define tableSize

Unsigned int hash(int key)

Make an insert method void HashTable::Insert(Course course)

Make key for specific course, search node for key value

IF no key entry

assign this node to the key position

ELSE if node is used

ALLOCATE the old node key to UNIT\_MAX, set to key, set old node to

course AND old node next to nullptr

ELSE find the next open node

add new newNode to end

void intakeCourses(string csvPath, HashTable\* hashTable)

LOOP to read rows of a CSV file

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

Create a data structure and add values

Course course;

course.courseId = file[i][1];

course.name = file[i][0];

while not end of line

course.prereq. = file[i][8];

hashTable->Insert(course);

BINARY TREE:

BinarySearchTree\* bst;

bst = new BinarySearchTree();

Course course;

Create add node method

void BinarySearchTree::addNode(Node\* node, Course course)

If root is null, add root

IF the node is less than root then add to left

IF no left node this node becomes left

IF the node is greater than root add right

IF no right node this node becomes right

void intakeCourses(string csvPath, BinarySearchTree\* bst)

LOOP through 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 courses

Course course;

course.courseId = file[i][1];

course.name = file[i][0];

while not end of line course.prereq. = file[i][8];

bst->Insert(course);

Print Course Information and Prerequisites:

VECTOR:

Create method void printCourseInfo(Vector courses, String courseId)

FIND input for courseId

WHILE the vector is not empty

IF the input is the same as courseId

Output course.courseId << output course.name

WHILE the (prereq = true)

Output course.prereq

HASHTABLE:   
Create the method

void printCourseInfo(Hashtable courses, String courseId)

FIND the input for courseId

Assign key = courseId

Assign node to the node.at(key)

IF current node matches key

Return course, displayCourse(nodes[key].course)

If node points to null, then return null

ELSE while the node is not Null, check against the key

If the key matches the couseId, Return course, displayCourse(nodes[key].course)

Point to next node

BINARY TREE:

Create method void printCourseInfo(Tree courses, String courseId)

Get input for courseId

Assign current node to root

While current is not NULL

If course.courseId matches current

Return current,

output course.courseId << output course.name

while (prereq = true)

out put course.prereq

If courseIid is less than root

Set current to left

Else set current to right

Menu: Set choice to 0;

Create while loop for menu.

While choice is not equal to 9

Output menu choices

(1. Load Course File,

2. Print Course

List 3. Print Individual Course

9.Exit)

Create switch(choice)

Case 1: intakeCourses(courseFile, structureData) Case 2: printSorted(courses) call function to print sorted class list

Case 3: printCourseInfo(courseId)

Case 9: Terminate Program

Print Sorted List:

//Vector

Create sorted print method printSorted(courses)

Create partition method int partition(vector& courses, int begin, int end)

Set lowIndex to first element, set highIndex to last element

Set midpoint to lowIndex + (highIndex - lowIndex) / 2

Set pivot to midpoint

Decrement highIndex while pivot is less than highIndex

Swap lower values to left of pivot, higher values to right of pivot

Set temp value to low index

Set low index to high index

Set high index to temp

Create quicksort method void quickSort(vector& courses, int begin, int end)

Set mid to 0, lowIndex to being, highIndex to end

If begin >= end,

return

Set lowEndIndex to partition(courses, lowIndex, highIndex)

Make recursive call to quicksort

quickSort(courses, lowIndex, lowEndIndex);

quickSort(courses, lowEndIndex + 1, highIndex)

Create display course method void displayCourse(Course course) {

cout << course.courseId << ": " << course.name << " | " << course.prereq << endl;

Loop through vector to display courses for (int i = 0; i < courses.size(); ++i) displayCourse(courses[i])//Tree

Create inOrder method void BinarySearchTree::inOrder(Node\* node)

If (node != Nul)

Check most left side first

inOrder(node->left)

cout << course.courseId << ": " << course.name << " | " << course.prereq << endl;

check next right leaf inOrder(node->right)

cout << course.courseId << ": " << course.name << " | " << course.prereq << endl;

Set low index to high index

Set high index to temp

Create quicksort method

void quickSort(vector& courses, int begin, int end)

Set mid to 0, lowIndex to being, highIndex to end

IF begin >= end,

return

Set lowEndIndex to partition(courses, lowIndex, highIndex)

Make recursive call to quicksort quickSort(courses, lowIndex, lowEndIndex); quickSort(courses, lowEndIndex + 1, highIndex)

Make the display course method

void displayCourse(Course course) {

cout << course.courseId << ": " << course.name << " | " << course.prereq << endl;

Loop through vector to display courses for (int i = 0; i < courses.size(); ++i) displayCourse(courses[i]))

Create inOrder method void BinarySearchTree::inOrder(Node\* node)

If (node != Null)

Check most left side first

inOrder(node->left)

cout << course.courseId << ": " << course.name << " | " << course.prereq << endl;

check next right leaf

inOrder(node->right)

cout << course.courseId << ": " << course.name << " | " << course.prereq << endl;

Runtime Analysis for Vector/Hashtable/BinaryTree

There are 6 lines in Vector. The total cost is 5n+1. The runtime is 0(n).

There are 17 lines for Hash table. The total cost is 16n+1. The runtime is 0(n).

There are 13 lines for Tree. The total cost is 11n+. The runtime is 0(n).

All the data structures have advantages and disadvantages to them. It relies heavily on the program needed and what data is being stored within it. Vectors have an advantage as they will look for a specific course with faster reading. But the disadvantage is that they must work through the entire structure till they find it. Hash tables can be complicated and involve a lot of information being stored within. They do create a key which helps find locations quicker. Hash Tables also have built-in collision operations. Some disadvantages of Hash tables are the limited space, and they do not maintain the order of the values, so it makes it more difficult to find. Binary Trees have Ordered traversal which helps to follow an ordered path through the values. They have quick insertion and deletions time complexity at 0(log n). Some disadvantages are the limited structure issues, unbalanced trees, and having issues with wors-case scenarios slowing processing.

I think a vector would be the best data structure for this project. The reason is that you can quickly sort to print the entire course catalog. The loss of time during the search is still worth being able to effectively find all the needed information in a less complicated and reasonable time. With a binary tree it could have off-balanced leaves, and the Hash table seems to require more information storing than needed.

References:

GeeksforGeeks. (2023a, March 28). *Applications, Advantages and Disadvantages of Hash Data Structure*. GeeksforGeeks. <https://www.geeksforgeeks.org/applications-advantages-and-disadvantages-of-hash-data-structure/>

GeeksforGeeks. (2023b, March 29). *Applications, Advantages and Disadvantages of Binary Tree*. GeeksforGeeks. <https://www.geeksforgeeks.org/applications-advantages-and-disadvantages-of-binary-tree/>

*Pseudocode for Binary search tree*. (n.d.). Stack Overflow. <https://stackoverflow.com/questions/29836015/pseudocode-for-binary-search-tree>

Olawanle, J. (2023, April 10). *Big o cheat sheet – Time Complexity chart*. freeCodeCamp.org. <https://www.freecodecamp.org/news/big-o-cheat-sheet-time-complexity-chart/>