Dylan Bishop

CS300

February 18, 2024

Project One Submission

**Open and read file pseudocode:**

Use fstream to open the file

Make a call to open the file, if the value returned is “-1”, then the file is not found.

Else the file is found

While it is not the end of the file

Read every line

If there are less than two values in one line

Return Error

Else read the parameters

If there is a third or more parameter

If the third or more parameter is in the first parameter elsewhere continue

Else

Return Error

Close the file

**Course Information:**

Create struct Course{}

Create variables: Course ID, Course Name, Prerequisite

**Creating Vector**

Vector<Course>loadCourses(string csvPath)

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

Creating structure of data and adding to the collection of the courses

Course course;

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

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

while it is not the end of the line

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

course.push\_back(course);

**Creating a Hash Table:**

Create a HashTable

Create Node struct

Course course

Unsigned int key

Vector<Node>nodes

Define the table size

Unsinged int has(int key)

Create the insert method void HashTable::Insert(Course course)

Create a key for the given course and search for the node that matches the key value

If there is no entry found for the key

Assign the node to the key position

Else if node is occupied

Assign the old node key to UNIT\_MAX, set it to the key and set the old node to course and old node next to nullptr

Else search for the next empty node

Add the newNode to the end

Void loadCourses(string csvPath, HashTable\*hashTable)

Create a loop to read the rows of a CSV file

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

Create a structure of data and add it to the collection of courses

Course course;

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

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

while it is not the end of the line

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

hashTable->Insert(course);

**Creating Binary Search Tree:**

Defining a Binary Search Tree so it can hold each course

BinarySearchTree\*BST;

BST = new BinarySearchTree();

Course course;

Create the add node method void BinarySearchTree::addNode(Node \*node, Course course)

If the root is null, then add a root

If the node is less than the root then add the node to the left

If there is no left node

Current node becomes the left node

If the node is greater than the root then add the node to the right

If there is no right node

Current node becomes the right node

Void loadCourses(string csvPath, BinarySearchTree\*BST)

Create a loop to read the rows of a CSV file

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

Create a structure of data and add it to the collection of courses

Course course;

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

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

while it is not the end of the line

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

BST->Insert(course);

**Printing all Course Info and Each course’s Prerequisites:**

**Printing Vector:**

Create method void printCourseInfo(Vector<Course> courses, String courseID)

Obtain the input for the courseID

While the vector is not empty

If input matches courseID

Print course.courseID << print course.courseName

While (preReq = true)

Print course.preReq

**Printing Hash Table:**

Create method void printCourseInfo(HashTable<Course> course, String courseID)

Obtain the input for the courseID

Assign the key to the courseID

Assign the node to the node.at(key)

If the current node matches the key

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

If the node points to null then return null

Else while node is != null, check if the node matches the key

If the key matches courseID then return course, displayCourse(nodes[key].course)

Point to the next node

**Printing the Binary Search Tree:**

Create method void printCourseInfo(Tree<Course> courses, String courseID)

Obtain the input for the courseID

Assign the current node to the root

While the current node is not NULL

If course.courseID matches the current node

Return current node and print course.courseID << print course.courseName

While (preReq = true)

Print course.preReq

If the courseID is less than the root

Set the current to the left

Else set the current to the right

**Menu Pseudocode:**

Set the choice to 0;

Create a while loop for the menu. While the choice is not equal to 4

Output the menu choices(1. Load the Course File 2. Print the Course List 3. Print the Individual Course 4. Exit Menu)

Create a switch(choice)

Case 1: loadCourses(courseFile, dataStructure) FIXME: use the structure of data structure chosen

Case 2: printSorted(courses) call the function to print the sorted class list

Case 3: printCourseInfo(courseID)

Case 4: Terminate the Program

**Printing the Sorted Course List:**

**Printing the Vector:**

Create a sorted print method printSorted(courses)

Create the partition method int partition(Vector<Course> & courses, int begin, int end)

Set the lowIndex to the first element and set the highIndex to the last element

Set the midpoint to the lowIndex + (highIndex – lowIndex) /2

Set the pivot to the midpoint

Decrement the highIndex while the pivot is less than the highIndex

Swap the lower values to the left of the pivot and the higher values to the right of the pivot.

Set a temp value to the low index

Set the low index to the high index

Set the high index to the temp value

Create the quicksort method void quickSort(vector<Course> & courses, int begin, int end)

Set the mid to 0, the lowIndex to the beginning, and the highIndex to the end

If the begin is >= the end, then return

Set the lowEndIndex to the partition(courses, lowIndex, highIndex)

Make a recursive call to quicksort

quickSort(courses, lowIndex, lowEndIndex)

quickSort(courses, lowEndIndex + 1, highIndex)

Create the display course method void displayCourse(Course course) {

Print <<course.courseID <<”: “ << courseName >> “ | “ << course.preReq << endline;

Loop through the vector to display the courses

For (int i =0; i < courses.size(); ++i)

displayCourse(courses[i])

**Printing the Binary Search Tree:**

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

If (node != null)

Check the left most side first

inOrder(node->left)

print << course.courseID << “: “ << course.courseName << “ | “ << course.preReq << endline;

Check the next right leaf

inOrder(node->right)

print << course.courseID << “: “ << course.courseName << “ | “ << course.preReq << endline;

**Runtime Analysis for Reading the CSV File and Creating the Course Objects**

|  |  |  |  |
| --- | --- | --- | --- |
| **Vector** | **Line Cost** | **Number of Executes** | **Total Cost** |
| Create the Vector | 1 | 1 | **1** |
| For each line in the file | 1 | n | n |
| Creating the vector course item | **1** | n | N |
| While a preReq exists | 1 | n | n |
| Append the preReq | 1 | n | n |
| Pushback the course item | 1 | N | N |
|  |  | **Total Cost** | **5n+1** |
|  |  | **Runtime** | **O(n)** |

|  |  |  |  |
| --- | --- | --- | --- |
| **HashTable** | **Line Cost** | **Number of Executes** | **Total Cost** |
| Create the hash table | 1 | 1 | 1 |
| Insert Method | 0 | 0 | 0 |
| Creating the key for course | 1 | n | n |
| If there is no entry found for key | 1 | n | n |
| Assign the node to the key | 1 | n | n |
| Else | 1 | n | n |
| Assign the old node key to UNIT\_MAX, set to key, set the old node to course and old node next to nullptr | 4 | n | 4n |
| Else | 1 | n | n |
| Find the next open node | 1 | n | n |
| Add newNode to the end | 1 | n | n |
| For each new line in the file | 1 | n | n |
| Create vector course item | 1 | n | n |
| While preReq exists | 1 | n | n |
| Append the preReq | 1 | n | n |
| Insert course item | 1 | n | n |
|  |  | **Total Cost** | **16n+1** |
|  |  | **Runtime** | **O(n)** |

|  |  |  |  |
| --- | --- | --- | --- |
| **Binary Search Tree** | **Line Cost** | **Number of Executes** | **Total Cost** |
| Add node method | 0 | 0 | 0 |
| If the root is null, add the root | 1 | 1 | 1 |
| If the node is less than the root, add it to the left | 1 | n | n |
| If there is no left node | 1 | n | n |
| Current node becomes the left | 1 | n | n |
| If the node is greater than the root, add it to the right | 1 | n | n |
| If there is no right node | 1 | n | n |
| Current node become the right | 1 | n | n |
| For each line in the file | 1 | n | n |
| Create vector course item | 1 | n | n |
| While preReq exists | 1 | n | n |
| Append the preReq | 1 | n | n |
| Insert course item | 1 | n | n |
|  |  | **Total Cost** | **11n + 2** |
|  |  | **Runtime** | **O(n)** |

For the requirements of this program, every different data structure has it’s own advantages and disadvantages. Using a vector has the disadvantage of having to search the entire list for one specific course until the match is found. The program will check every single item, however, using the vector method is the fastest method that will read the file and add the course objects. The runtime of the vector method was the shortest of the three with a runtime of 5n+1.

Using a hash table has the advantage of searching a list quickly with creating a key because the locations can easily be searched and printed. Although, this method is much slower when creating the list because it needs to find a spot for each course to be inserted. Sorting the Hash table is not possible within itself. You must extract each value, sort it, and then you are able to print an alphanumeric list of each course. This method had the slowest runtime of 16n+1.

The Binary Search Tree is better than using a vector because the Binary Search Tree has the quickest ability to sort. It is not easier than using a Hash Table but it is quicker to use the Binary Search Tree over the vector. The search time of the Binary Search Tree is O(h), h being the height of the tree. The Binary Search Tree Method is the second quickest runtime with a runtime of 11n+2.

I would personally recommend using the vector sort method for this project because being able to sort the list quickly and print all its entire list is much more valuable. The runtime is the quickest and losing time during the search is not as bad as the utility of the sort.