Douglas Bowers

CS-300

4/11/2024

6-2 Project One

Reading File:

Use fstream to open file

Create method void loadCourses(string csvPath, dataStructure)

Make call to open file, if return value is -1 then file not found

Else file is found

While it is not EOF

Read each line

If there are less than two values in a line, return Error

Else read parameters

If there are three of more parameters

If 3 or more parameters are in first parameter elsewhere, continue

Else return Error

Close file

Hold Course Information:

Create struct Course{}

Create Identifiers: Course ID, Course Name, Prerequisite

Vector:

vector<Course>loadCourses(string csvPath)

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

Create data structure, add to course collection

Course course;

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

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

While not EOF

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

Course.push\_back(course);

Hash Table:

Create Hashtable

Create node struct

Course course

Unisgned int key

Vector<Node>nodes

Define tableSize

Unsigned int had(int key)

Create insert method void HashTable::Insert(Crouse course)

Create key for given course, search for node with that key value

If no key value found

Assign this node to key position

Else if node is used

Assign old node key to UNIT\_MAX, set to key, set old node to course and old node next to null pointer

Else find the next open node

Add new newNode to end

Void loadCourses(string csvPath, HashTable\*hashtable)

Loop to read rows from CSV file

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

Create data structure and add to course collection

Course course;

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

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

While not EOF

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

hashTable->Insert(course);

Tree:

Define binary search tree to hold all courses

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 node is less than root, add to left

If no left node

Node becomes left

If node is greater than root, add to right

If node right node

Node becomes right

Void loadCourses(string csvPath, BinarySearchTree\*bst)

Loop to read rows from CSV file

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

Create data structure and add to course collection

Course course;

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

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

While not EOF

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

Bst->Insert(course);

Print Course Information and Prerequisites:

Vector//

Create method void printCourseInformation(Vector<Course> courses, String courseId)

Get input for courseId

While vector is not empty

If the input is the same as courseID

Output course.courseID << output course.name

While (prereq = true)

Output course.prereq

HashTable//

Create method void printCourseInformation(Hashtable<Course> courses, String courseId)

Get 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, return null

Else while node is not null, check against key

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

Point to next node

Tree//

Create method void printCourseInformaiton(Tree<Course> courses, String courseId)

Get input for courseId

Assign current node to root

While current node is not Null

If course.courseId matches current node

Return current node, output course.courseId << output course.anem

While (prereq = true)

Output course.prereq

If courseId 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 4

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

Create switch(choice)

Case 1: loadCourses(courseFile, dataStrucutre) Fixme: use structure of data structure chosen

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

Case 3: printCourseInformation(courseId)

Case 4: Exit Program

Print Sorted List:

Vector//

Create sorted print method printSorted(courses)

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

Set mid to 0, lowIndex to begin, 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 != 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:

|  |  |  |  |
| --- | --- | --- | --- |
| Vector | Line Cost | # Times Executed | Total Cost |
| Create vector | 1 | 1 | 1 |
| For each line in file | 1 | n | n |
| Create vector course item | 1 | n | n |
| Create vector | 1 | 1 | 1 |
| While prereq exists | 1 | n | n |
| Append prereq | 1 | n | n |
| Pushback course item | 1 | n | n |
|  |  | Total Cost | 5n+1 |
|  |  | Runtime | O(n) |

|  |  |  |  |
| --- | --- | --- | --- |
| HashTable | Line Cost | # Times executed | Total Cost |
| Create Hashtable | 1 | 1 | 1 |
| Insert method | 0 | 0 | 0 |
| Create key for course | 1 | n | n |
| If no entry found for key | 1 | n | n |
| Assign node to key | 1 | n | n |
| else | 1 | n | n |
| Assign old node key to UNIT\_MAX, set old node to course and old node next to null pointer | 4 | n | 4n |
| else | 1 | n | n |
| Find next open node | 1 | n | n |
| Add new newNode to end | 1 | n | n |
| For each line in file | 1 | n | n |
| Create vector course item | 1 | n | n |
| While prereq exists | 1 | n | n |
| Append prereq | 1 | n | n |
| Insert course item | 1 | n | n |
|  |  | Total Cost | 16n+1 |
|  |  | Runtime | O(n) |

|  |  |  |  |
| --- | --- | --- | --- |
| Tree | Line Cost | # Times Executed | Total Cost |
| Create tree | 1 | 1 | 1 |
| Add node method | 0 | 0 | 0 |
| If root is null, add root | 1 | 1 | 1 |
| If node is less than root, add to left | 1 | n | n |
| If no left node | 1 | n | n |
| Node becomes left | 1 | n | n |
| If node is greater than root, add to right | 1 | n | n |
| If no right node | 1 | n | n |
| Node becomes right | 1 | n | n |
| For each line in file | 1 | n | n |
| Create vector course item | 1 | n | n |
| While prereq exists | 1 | n | n |
| Append prereq | 1 | n | n |
| Insert course item | 1 | n | n |
|  |  | Total Cost | 11n+2 |
|  |  | Runtime | O(n) |

One advantage of the vector method is that it is the fastest method for reading a file and adding course objects. As the files is parsed, each item is appended to the end of a vector, making this a very straightforward method. Out of the three methods, the runtime was the shortest at 5n+1, however, they all shared the same notation of O(n). A disadvantage of the vector method would be searching the list for a specific course. The program must check each item within the vector until a match is found.

One advantage of hash tables is that you can quickly search a list. After creating a key, the locations of a given course will be known, and it can easily be searched and printed. However, it is a slower implementation when creating the initial list, as each item needs a respective key given to it, and a spot needs to be found to insert each course. Hash tables also do not lend themselves to sorting, thus the table itself cannot be sorted. To print an alphanumerical list, each value must be extracted, sorted, and then printed.

One advantage of Binary tree is that they are faster to search than the vector method. When the course being searched for is known, it is easy to run down the tree until the value is found. However, it is not as easy as a hash table, but it is faster than the vector method. The tree may have to search every element if the tree ended up with only left leaves, thus putting the search time at O(h), where h is the height of the tree.

For this project, I would recommend the vector sort method. Being able to quickly sort to print the entire catalog would be valuable to a client so they are able to see each item and its value. While there is a loss of time when searching, the utility that it provides would be more beneficial.