Molly Vaughns

CS-300: DSA: Analysis and Design

Ricardo Scarello

February 18, 2024

**Module 6 Assignment: Project 1**

**Pseudocode:**

**Reading file:**

USE fstream to be able to open file

CREATE method void loadCourses(string csvPath, dataStructure)

CALL to open file, IF return vlue is “-1”, file is not found

ELSE file is found

WHILE it is not end of file

READ each line

IF less than two values in line, RETURN error

ELSE read parameters

IF third or more parameters

IF third or more parameter is 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 and add to collection of courses

Course course;

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

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

WHILE not end of line

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

Courses.push\_back(course);

**//Hash Table**

CREATE hashtable

CREATE node struct

Course course

Unsigned int key

Vector<node>nodes

DEFINE tableSize

Unsigned int has(int key)

CREATE insert method void HashTable::Insert(Course course)

CREATE key for given course, SEARCH for node with key value

IF no entry found for key

ASSIGN 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 next open node

ADD new newNode to end

Void loadCourses(string csvPath, HashTable\* hashTable)

LOOP to read rows of CSV file

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

CREATE data structure and add to 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];

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, then add to left

IF no left node

Node becomes left

IF node is greater than root, then add to right

IF no right node

Node becomes right

Void loadCourse(string csvPath, BinarySearchTree\* bst)

LOOP to read rows of CSV file

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

CREATE data structure and add to 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 printCourseInformation(Vector<Course> courses, String courseId)

ASK for input of courseId

WHILE vector is not empty

IF input is same as courseId

OUTPUT course.courseId<<coutput course.name

WHILE (prereq = true)

OUTPUT course.prereq

**//Hash Table**

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

ASK for input of courseId

ASSIGN key = courseId

ASSIGN node to 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 key matches courseId, RETURN course, dispayCourse(nodes[key].course)

POINT to next node

**//Tree**

CREATE method voice printCourseInformation(Tree<Course> courses, String courseId)

ASK for input of 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)

OUTPUT course.prereq

IF courseId 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 equa 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, dataStructure) use structure of data structure chosen

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

CASE 3: printCourseInformation(courseId)

CASE 4: Terminate 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 vaues to eft of pivot, higher values to right of pivot

SET temp value to lowIndex

SET lowIndex to highIndex

SET highIndex to temp

CREATE quicksort method void quicksort(vector<Course>& courses, int begin, int end)

SET mid to 0; lowIndex to being, highIndex to end

IF begin >= rend, RETURN

SET lowEndIndex to partition(courses, lowIndex, highIndex)

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 <course.size(); i++)

displayCourse(courses[i])

**//Tree**

CREATE inOrder method voice inarySearchTree::inOrder(Node\* node)

IF (node != null)

CHECK most left side first

inOrder(node->eft)

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 Reading the File and Creating Course Objects:**

|  |  |  |  |
| --- | --- | --- | --- |
| Vector | Line Cost | # Times Executes | Total Cost |
| Create Vector | 1 | 1 | 1 |
| 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 |
| Pushback course item | 1 | N | N |
| Total Cost | | | 5n+1 |
| Runtime | | | O(n) |

|  |  |  |  |
| --- | --- | --- | --- |
| HashTable | Line Cost | # Times Executes | Total Cost |
| Create hash table | 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 to key, set old node to course and old node next to null pointer | 4 | n | 4n |
| Else | 1 | n | n |
| Find the next open node | 1 | n | n |
| Add new newNode to end | 1 | n | n |
| For each new 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 Executes | Total Cost |
| Add node method | 0 | 0 | 0 |
| If root is null, add root | 1 | 1 | 1 |
| If node is less than root, then add to left | 1 | n | n |
| If no left node | 1 | n | n |
| This 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 |
| This 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+1 |
| Runtime | | | O(n) |