Elizabeth Robles

CS-300 DSA: Analysis and Design

Southern New Hampshire University

October 7th, 2022

Project One

**Vector Pseudocode**

***Define a structure to hold course information***

String for courseNumber

String for courseName

Vector <String> for prerequisites

***Read data from file, parse each line, and check for errors***

Create loadCourses method to load a csv file containing courses to a container

Define a vector data structure to hold a collection of courses, vector<Course>courses

Initialize the csv parser using csvPath

Create a try catch loop

Use ifstream to be able to open the file in the csvpath

If the courses file does open

While not the end of the file

Create a vector to hold csv

While course data is greater than 0

Find the commas in the file

If comma

Erase the comma

Else

Add the very last of the courses after the comma

Create a data structure and add to the collection of courses

Course course

course.courseNumber = file[i][0]

course.courseName = file[i][1]

If there are > 2 parameters

course.prerequisites = file[i][2]

push this course to the end

catch any errors

return courses

***Print course information and prerequisites for searched course***

Create a method for void printCourseInformation(Vector<Course>courses, String courseNumber)

Get the users input for courseNumber

For all courses available

If the course is the same as courseNumber listed in database

Display the courseNumber and courseName to screen

For each valid prerequisite for the courses

Display the prerequisite information to the screen

***Print out the list of the courses in the CS program in alphanumeric order***

Partition the vector of courses into 2 parts low and high

Set low and high equal to begin and end

Pick the middle element as a pivot point

While not done

While loop to increment low index while courses[low] is less than courses [pivot]

While loop to decrement high index while courses[pivot] is less than courses [high]

If there are 0 or one elements remaining, all courses partitioned

Return high

Else

Swap the low and high courses

Move low and high closer

Perform a quickSort on course title

Declare int mid equal to o

If there are 1 or zero courses to sort, partition is already sorted

If begin is greater than or equal to end

Return

Partition courses into low and high such that the midpoint is the location of the last element in low

Sort low partition begin to mid

Sort high partition mid plus 1 to end

***Display a menu to user***

In main method

Command line arguments

Define a vector to hold all courses vector<Course>courses

Define timer variable

Declare int choice to 0

While the choice does not equal 4 execute menu

Display “Menu” to screen

Display “1. Load Courses” to screen

Display “2. Quick Sort all Courses” to screen

Display “3. Print Course Information" to screen

Display “4. Exit” to screen

Ask user to enter their choice

Create switch statement for the users choice option

Case 1:

Initialize timer variable before loading all courses

Complete the method call to loadCourses

Display the number of courses read to screen

Then calculate the elapsed time and display the result on screen

Break

Case 2:

Initialize timer variable

Call the quickSort method

displayCourses

Break

Case 3:

Call the printCourseInformation method

Display “Goodbye.” to screen

Return 0

**Hash Table Pseudocode**

***Define a structure to hold course information***

String for courseNumber

String for courseName

Vector <String> for prerequisites

***Read data from file, parse each line, and check for errors***

Create loadCourses method to load a csv file containing courses to a container using HashTable

Initialize the csv parser using csvPath

Create a try catch loop

Use ifstream to be able to open the file in the csvpath

If the courses file does open

While not the end of the file

Create a vector to hold csv

While course data is greater than 0

Find the commas in the file

If comma

Erase the comma

Else

Add the very last of the courses after the comma

Create a data structure and add to the collection of courses

Course course

course.courseNumber = file[i][0]

course.courseName = file[i][1]

If there are > 2 parameters

course.prerequisites = file[i][2]

push this course to the end

catch any errors

***Print course information and prerequisites for searched course***

Create a method for void printCourseInformation(Hashtable<Course>courses, String courseNumber)

Create a key for the given bid

Then access the course within the nodes hash table list for specific course

For all courses available

If the course is the same as courseNumber listed in database

Display the courseNumber and courseName to screen

For each valid prerequisite for the courses

Display the prerequisite information to the screen

***Print out the list of the courses in the CS program***

Create printSampleSchedule method

For node begin to end iterate

If the key is not equal to UINT\_MAX

Display key, course information, and prerequisites

Node is equal to the next iteration

While node is not empty then

Display key, course information, and prerequisites

Node is equal to the next node

***Pseudocode for a menu***

In main method

Command line arguments

Define timer variable

Define a hash table to hold all the courses

Course course

courseTable is equal to new hash table

Declare int choice to 0

While the choice does not equal 4 execute menu

Display “Menu” to screen

Display “1. Load Courses” to screen

Display “2. Display all Courses” to screen

Display “3. Print Course Information" to screen

Display “4. Exit” to screen

Ask user to enter their choice

Create switch statement for the users choice option

Case 1:

Initialize timer variable before loading all courses

Complete the method call to load the courses

Display the number of courses read to screen

Then calculate the elapsed time and display the result on screen

Break

Case 2:

Course table points to PrintAll method

Break

Case 3:

Point to the printCourseInformation method

Break

Display “Goodbye.” to screen

Return 0

**Tree Pseudocode**

***Define a structure to hold course information***

String for course number

String for course name

Vector <String> for prerequisites

***Read data from file, parse each line, and check for errors***

Create loadCourses method to load a csv file containing courses to a container using BST

Initialize the csv parser using csvPath

Create a try catch loop

Use ifstream to be able to open the file in the csvpath

If the courses file does open

While not the end of the file

Create a vector to hold csv

While course data is greater than 0

Find the commas in the file

If comma

Erase the comma

Else

Add the very last of the courses after the comma

Create a data structure and add to the collection of courses

Course course

course.courseNumber = file[i][0]

course.courseName = file[i][1]

If there are > 2 parameters

course.prerequisites = file[i][2]

push this course to the end

catch any errors

***Print course information and prerequisites for searched course***

Search for a course by creating a printCourseInormation method

While current node does not equal nullptr

Create if loop to loop downwards until bottom is reach or a matching courseNumber is found

If match is found

Return the current course information to screen

Check to see if there are any prerequisites

If a prerequisite does exist

Display all prerequisites to the screen

Else

If they do not exist then tell user that there are no prerequisites for that course

Return

***Print out the list of the courses in the CS program in alphanumeric order***

Traverse the tree in order by creating a PrintSampleSchedule method

Call the printSampleSchedule function and pass the root

Create an printSampleSchedule function

If node is equal to nullptr

Return

InOrder not left

Display courseNumber, courseName, and prerequisites to screen

InOrder right

***Pseudocode for a menu***

In main method

Command line arguments

Define timer variable

Define a binary search tree to hold all of the courses

bst is equal to new Binary search tree

Declare int choice to 0

While the choice does not equal 4 execute menu

Display “Menu” to screen

Display “1. Load Courses” to screen

Display “2. Display all Courses” to screen

Display “3. Print Course Information" to screen

Display “4. Exit” to screen

Ask user to enter their choice

Create switch statement for the users choice option

Case 1:

Initialize timer variable before loading all courses

Complete the method call to load the courses

Display the number of courses read to screen

Then calculate the elapsed time and display the result on screen

Break

Case 2:

bst points to the InOrder method to print all courses in order

Break

Case 3:

Call search method for courseKey

If course exists

displayCourse is called then displays course and course info

Else

Course not found

Break

Display “Goodbye.” to screen

Return 0

**Worst case runtime for Vector**

|  |  |  |  |
| --- | --- | --- | --- |
| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| Vector<Course>loadCourse(string csvPath) | 0 | 0 | 0 |
| Define a vector data structure to hold courses | 1 | 1 | 1 |
| Initialize csv parser using the csvPath | 1 | 1 | 1 |
| Try catch loop | 1 | n | n |
| Use ifstream to be able to open file in csv path | 1 | 1 | 1 |
| If the courses file does open | 1 | n | n |
| While not the end of file | 1 | n | n |
| Create vector to hold csv file | 1 | n | n |
| While course data is > 0 | 1 | n | n |
| Find commas | 1 | n | n |
| If comma | 1 | n | n |
| Erase comma | 1 | n | n |
| Else | 1 | n | n |
| Add the very last of the courses after comma | 1 | n | n |
|  |  |  |  |
| Create data structure to add to collection of courses | 0 | 0 | 0 |
| Course course | 1 | n | n |
| course.courseNumber = file[i][1] | 1 | n | n |
| course.courseName = file[i][0] | 1 | n | n |
| If there are > 2 parameters | 1 | n | n |
| course.prerequisites = file[i][8] | 1 | n | n |
| push this course to the end | 1 | n | n |
| catch any errors | 1 | n | n |
|  |  | **Total Cost** | 17n+3 |
|  |  | **Runtime** | O(n) |

**Worst case runtime for Hash Table**

|  |  |  |  |
| --- | --- | --- | --- |
| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| Void loadCourses(string csvPath, HashTable\* hashtable) | 0 | 0 | 0 |
| Initialize csv parser using csvPath | 1 | 1 | 1 |
| Try catch loop | 1 | n | n |
| Use ifstream to be able to open file in csv path | 1 | 1 | 1 |
| If the courses file does open | 1 | n | n |
| While not the end of file | 1 | n | n |
| Create vector to hold csv file | 1 | n | n |
| While course data is > 0 | 1 | n | n |
| Find commas | 1 | n | n |
| If comma | 1 | n | n |
| Erase comma | 1 | n | n |
| Else | 1 | n | n |
| Add the very last of the courses after comma | 1 | n | n |
|  |  |  |  |
| Create data structure to add to collection of courses | 0 | 0 | 0 |
| Course course | 1 | n | n |
| course.courseNumber = file[i][1] | 1 | n | n |
| course.courseName = file[i][0] | 1 | n | n |
| If there are > 2 parameters | 1 | n | n |
| course.prerequisites = file[i][8] | 1 | n | n |
| push this course to the end | 1 | n | n |
| catch any errors | 1 | n | n |
|  |  | **Total Cost** | 17n+2 |
|  |  | **Runtime** | O(n) |

**Worst case runtime for BST**

|  |  |  |  |
| --- | --- | --- | --- |
| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| Void loadCourses(string csvPath, BinarySearchTree\* bst) | 0 | 0 | 0 |
| Initialize csv parser using csvPath | 1 | 1 | 1 |
| Try catch loop | 1 | n | n |
| Use ifstream to be able to open file in csv path | 1 | 1 | 1 |
| If the courses file does open | 1 | n | n |
| While not the end of file | 1 | n | n |
| Create vector to hold csv file | 1 | n | n |
| While course data is > 0 | 1 | n | n |
| Find commas | 1 | n | n |
| If comma | 1 | n | n |
| Erase comma | 1 | n | n |
| Else | 1 | n | n |
| Add the very last of the courses after comma | 1 | n | n |
|  |  |  |  |
| Create data structure to add to collection of courses | 0 | 0 | 0 |
| Course course | 1 | n | n |
| course.courseNumber = file[i][1] | 1 | n | n |
| course.courseName = file[i][0] | 1 | n | n |
| If there are > 2 parameters | 1 | n | n |
| course.prerequisites = file[i][8] | 1 | n | n |
| push this course to the end | 1 | n | n |
| catch any errors | 1 | n | n |
|  |  | **Total Cost** | 17n+2 |
|  |  | **Runtime** | O(n) |

**Advantages and Disadvantages**

There are a few advantages and disadvantages to using either the vector, hash table, or tree data structure as they all have their own set of pros and cons depending on what they need to be used for. A vector would be best suited for a program that requires constant resizing, in other words, a program that needs data to be constantly added and deleted, with deleting being easier and faster as it operates in constant time, adding data however, could take longer. A hash table is a data structure in which a table(array or vector) is set up from 0 to however many items needed, and each value is mapped into a specific location inside said table using the modulo computation, these values are also known as keys. This data structure is good for searching, inserting, and removing items from the array or vector, and only requires 0(1) for insertions, deletions, and searches. However, if the table becomes too full, collisions can occur and will require for chaining techniques to fix the issue. Lastly, a binary search tree is a data structure in which nodes are placed in a tree, with the led subtree nodes being less than the root, and the right subtree nodes being greater than the root, this allows for very fast searching of a specific value.

**Evaluation**

Overall, all the different options would work well for the program ABCU wants us to create. Since we do not need to prioritize insertions or deletions to the data structure, as the size of the data will not change, I will go with a binary search tree for this program. This will easily allow me to create a method that will print a list of all the Computer Science courses in alphanumeric order, as well as quickly search for and print out a specific courses title and prerequisite information.