Kerrian Offermann

CS-300 Module Six

6-2 Assignment: Project One

April 8, 2022

**Resubmitted: Pseudocodes for Vector, Hash Table, and Tree**

* **Vector Pseudocode**

**Open File, Read Data, Parse Lines, and Check Formatting Errors**

vector<Course>courses(string csvpath)

Initialize “vector<Course> courses” to hold data from file

Use a csv parser to read through each line

Use ifstream to open csv file

If the csv file is open

While it is not the end of the file

Set vector string to courseInfo

Set courseData to string

Get line of course file

While the length of data is not zero

If there is data then add

Separate data based on commas

Insert into vector

Sort based on course number

Else

If there is a prerequisite course

Insert after main course

Set up try-catch to locate errors in the file

For all lines in the csv file

Add each line of courses to the vector

Parse each based on the comma delimiter to separate

The first token is assigned to variable courseNumber

The second token is assigned to courseTitle

The third token is assigned to coursePrerequisite

End try-catch to locate errors

Return

**Create Course Objects – One Object Holds One Line of Data**

void printSchedule(Vector<Course> courses)

Initialize string csv path

For all courses in vector

Print course number

Print course title

If course being iterated has a prerequisite line

Print course number of prerequisite

Print course title of prerequisite

**Print Course Information and Prerequisites**

void printCourseInformation(Vector<Course> courses, String courseNumber)

For all courses in vector

If the course being iterated is the same as courseNumber

Print out the course information

For each prerequisite of the course

Print the prerequisite course information

* **Hash Table Pseudocode** **Open File, Read Data, Parse Lines, and Check Formatting Errors**

Int numPrerequisiteCourses(HashTable<Course> courses)

Initialize csv parser

Use ifstream to open csv file

If the csv file is open

While it is not the end of the file

Set vector string to courseInfo

Set courseData to string

Get line of course file

While the length of data is not zero

If there is data then add

Separate data based on commas

Create a bucket for each data file

Else

If there is a prerequisite course

Add to same bucket as main course

Initialize HashTable

Set csvPath as a string

Set key as a string and courseNumber

Set try and catch

For (i = 0; i less than file’s row count; iterate with i++)

If searching HashTable with key brings back null

Set courseList to equal HashTable[Hash(course->key)

Allocate new node

Set node->next to null

Set node->course to courseKey

ListAppend(courseList, node)

**Create Course Objects – One Object Holds One Line of Data**

void printSampleSched (HashTable<Course> courses)

   For each key in HashTable[courses]

Iterate through HashTable[courses]

Print HashTable[courses] value associated with the key

If HashTable[courses] key has prerequisite course

For key and value of prerequisite course in HashTable[courses]  
                   Print course from HashTable[courses] with prerequisite key and value

**Print Course Information and Prerequisites**

void printCourseInfo (HashTable<Course> courses, String courseNumber)

For courses in HashTable[courses]

If the course in HashTable[courses] equals key and value of courseNumber

Print course information for HashTable[courses]

For each key and value of prerequisite course in HashTable[courses]

Print course information for prerequisite course

* **Tree Pseudocode**

**Open File, Read Data, Parse Lines, and Check Formatting Errors**

Initialize courses to equal courses in Tree<Course>

For every line in the csv file

Use ifstream to open csv file

If the csv file is open

While it is not the end of the file

Set vector string to courseInfo

Set courseData to string

Get line of course file

While the length of data is not zero

If there is data then add

Separate data based on commas

Add to current node

If less than current node move left

If more than current node move right

Else

If there is a prerequisite course

Add as a child of main course

Set try and catch

Set root to first value of courseName (CSCI100)

While the current node is not null

For each course in tree of courses

Add courseName as left child of the root

If courseName greater than the left child

Add courseName as a right child

If course is a prerequisite (prereq)

Add prerequisite (prereq) as a left child of courseName

If current iteration is already present in tree

Do not add to tree again

**Create Course Objects – One Object Holds One Line of Data**

void printSampleSchedule(Tree<Course> courses)

For each node in Tree<Course>

Print root

While current node is not null

For all nodes in Tree<Course>

Print the courseName

If the courseName has nodes/children

Print the children starting with left child

Else

Print the courseName

**Print Course Information and Prerequisites**

void printCourseInformation(Tree<Course> courses, String courseNumber)

While current node is not null

For all nodes in Tree<Course>

Print courseName, courseNumber, and courseInformation in current node

If current node has a left child

Print prerequisite in left node

Print course info of prerequisite in left node

If current node has a right child

Print prerequisite in right node

Print course info of prerequisite in right node

**Pseudocode for Menu**

Create integer method for main menu

Initialize string for csv path

Initialize string for coursed

Initialize input variable “option” as ‘0’

Output: “Welcome to the course planner!”

Output: “Please select an option from the menu”

Allow user to enter an input

While the option is not 9

Output: “1. Load Data Structure.”

Output: “2. Print Course List.”

Output: “3. Print Course.”

Output: “9. Exit.”

If user enters an option that is not 1, 2, 3, or 9

Output: “Sorry, but [input] is not a valid option. Please try again.”

Return user to the main menu

If user enters option 1

Load the chosen data structure and csv list

If user enters option 2

Output: “Here is a sample schedule”

Print the course list

Return user to the main menu

If the user enters option 3

Output: “What course would you like information about?”

Input: Allow user to enter course

If the course is valid

Print course name, “,”, and course description

Print ‘Prerequisite:’ and prerequisite course

If there are more than one prerequisite course

Print “,” then SPACE then course name

Return to the main menu

Else

Output: “That course does not exist. Please try again.”

Return user to the main menu.

If the user enters option 4

Output: “Thank you for using the course planner. Goodbye!”

**Pseudocode Courses Printed in Alphanumeric Order**

* **Vector**

Initialize “vector<Course> courses” to hold data from file

Use a csv parser to read through each line

Add each line of courses to the vector

Parse each based on the comma delimiter to separate

The first token is assigned to variable courseNumber

The second token is assigned to courseTitle

The third token is assigned to coursePrerequisite

For (i = 1; i < vector size; ++i)

Initialize variable j to equal iteration i

While j is greater than zero and courses[j] is less than courses [j-1]

Swap courses[j] and courses[j-1]

Set courses[j] to variable “temp”

Set courses[j] equal to courses[j-1]

Set courses[j-1] equal to variable “temp”

For all courses in vector

Print course number

Print course title

If course being iterated has a prerequisite line

Print course number of prerequisite

Print course title of prerequisite

* **Hash Table**

Int numPrerequisiteCourses(HashTable<Course> courses)

Initialize csv parser

Initialize HashTable = Course(course.key)

Set csvPath as a string

Set key as a string and courseNumber

coursesSearched = 0;

While coursesSearched is less than HashTable size

If the hash table’s current bucket is empty

Add row of data to csvPath

Iterate through the number of buckets

If the hash table’s current bucket has data

Add to another bucket

For courses in HashTable[courses]

If the course in HashTable[courses] equals key and value of courseNumber

Print course information for HashTable[courses]

For each key and value of prerequisite course in HashTable[courses]

Print course information for prerequisite course

* **Tree**

Initialize courses to equal courses in Tree<Course>

For every line in the csv file

Add first value as courseNumber

Add second value (after comma delimiter) as courseName

Add every value after second value as prereq

Set root to first value of courseName (CSCI100)

While the current node is not null

For each course in tree of courses

Add courseName as left child of the root

If courseName greater than the left child

Add courseName as a right child

If course is a prerequisite

Add prerequisite as a left child of courseName

If current iteration is already present in tree

Do not add to tree again

For all nodes in Tree<Course>

Print courseName, courseNumber, and courseInformation in current node

If current node has a left child

Print prerequisite in left node

Print course info of prerequisite in left node

If current node has a right child

Print prerequisite in right node

Print course info of prerequisite in right node

**Evaluation**

| **Vector Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **Open the file** | 2 | n | 2n |
| **Read data from the file** | 1 | n | n |
| **Parse through each line** | 24 | 1 | 24 |
| **Check formatting errors** | 3 | n | n |
| **Create course objects** | 24 | n | 24n |
| **Total Cost** | | | 28n + 24 |
| **Runtime** | | | O(n) |

| **Hash Table Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **Open the file** | 2 | n | 2n |
| **Read data from the file** | 1 | n | n |
| **Parse through each line** | 8 | 1 | 8 |
| **Check formatting errors** | 3 | n | n |
| **Create course objects** | 8 | n | 8n |
| **Total Cost** | | | 12n + 8 |
| **Runtime** | | | O(1) |

| **Tree Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **Open the file** | 2 | n | 2n |
| **Read data from the file** | 1 | n | n |
| **Parse through each line** | 8 | 1 | 8 |
| **Check formatting errors** | 3 | n | n |
| **Create course objects** | 8 | n | 8log(n) |
| **Total Cost** | | | 4n + 8log(n) + n |
| **Runtime** | | | O(log(n)) |

**Data Structures: Advantages, Disadvantages, and Recommendations**

**Vector:** The greatest advantage of vectors is its simplicity. Especially for beginners, the concept of vectors is easier to understand since it is linear in the way it works. This is why adding a new element to a vector involves less coding than hash tables or trees. However, when it comes to vectors, it is difficult to insert and remove elements without iterating to specific positions. This can be consuming in terms of time and memory.

**Hash Tables:** Hash tables have the ability to make pair data with different keys. Because of this, it is easier to locate data via keys, and this it is easier to insert, add, or delete elements as needed. It is also far quicker to search for data. Disadvantages of using hash tables include having to set the size of the table beforehand and handling collisions when dealing with keys of the same value. These tasks are challenging to understand as well as implement despite the many advantages that hash tables provide.

**Trees:** Trees come with the major advantage of organizing itself which means that finding and searching for data is both faster and easier. It is also simple to add data since parent nodes serve as a guide to where data should go, and the child nodes provide more details about what falls into the parent node’s categories. In terms of disadvantages, trees require navigating through the tree in order to insert and delete nodes, so a tree with a large height could lead to complexities when adding/removing data.

**Recommendation:** When it comes to a data structure that can best handle storing information about courses, showing their relationship with other courses (via prerequisites), and adding new courses in an organized fashion then the choice seems obvious: **trees**. Trees are not only fast in terms of searching for data (based on Big O analysis), but by simply looking at the flowchart of courses we can see that a tree structure would be ideal for iterating through courses and printing their information. Courses, prerequisites, and their information can all easily be connected through different parent-child relationships and then accessed as needed with each search.