Shane Doyle 2/10/23

Professor Thoma Project One Pseudocode

**Vector**

File Input

Declare an ifstream object inputFile

Open the input file

IF input file successfully opened

WHILE EOF is not reached

IF less than 2 parameters exist on a line

Return file format error

ELIF 2 parameters exist on line

READ in course number and course title values

ELSE

READ in course number, course title, and prerequisite values

IF prerequisite value(s) don’t exist in course file

Return file format error

Close file

Course Object Pseudocode

Declare new Course object

Declare integer courseNumber

Declare string courseTitle

Declare vector <string> prerequistes

WHILE EOF is not reached

READ in courseNumber

READ in courseTitle

READ in prerequisites if any exist

Store courseNumber, courseTitle, and any prerequisites in course object

PUSH course object onto courses vector

Print Course Information Pseudocode

Prompt user for courseNumber

INPUT courseNumber

Call printCourseInformation

For all courses

IF the course is the same as the course number

OUTPUT course information

For each course prerequisite

OUTPUT the prerequisite course information

**Hash Table**

File Input

Declare an ifstream object inputFile

Open the input file

IF input file successfully opened

WHILE EOF is not reached

IF less than 2 parameters exist on a line

Return file format error

ELIF 2 parameters exist on line

READ in course number and course title values

ELSE

READ in course number, course title, and prerequisite values

IF prerequisite value(s) don’t exist in course file

Return file format error

Close file

Course Object Pseudocode

Declare new Hash Table object

Declare new Course object

Declare integer courseNumber

Declare string courseTitle

Declare vector <string> prerequistes

WHILE EOF is not reached

READ in courseNumber

READ in courseTitle

READ in prerequisites if any exist

Store courseNumber, courseTitle, and any prerequisites in course object

Insert Course object into Hash Table

Find Number of Prerequisites Pseudocode

Function numPrerequisiteCourses(Hashtable<Course> courses)

Find key value from bid

Find node using key

While node does NOT equal nullptr

IF node courseNumber equals courseNumber

IF course has course prerequisites

FOR each prerequisite p in totalPrerequisites

add prerequisites of p to totalPrerequisites

Return number of totalPrerequisites

Print Course Information Pseudocode

Function printCourseInformation(Hashtable<Course> courses, String courseNumber)

Find key value from courseNumber

Find node using the key

While node does NOT equal nullptr

IF node courseNumber equals courseNumber

OUTPUT course information

For each course prerequisite

OUTPUT the prerequisite course information

**Binary Search Tree**

File Input

Declare an ifstream object inputFile

Open the input file

IF input file successfully opened

WHILE EOF is not reached

IF less than 2 parameters exist on a line

Return file format error

ELIF 2 parameters exist on line

READ in course number and course title values

ELSE

READ in course number, course title, and prerequisite values

IF prerequisite value(s) don’t exist in course file

Return file format error

Close file

Course Object Pseudocode

Declare new Binary Search Tree object bst

Declare root equal to nullptr

Declare new Course object course

Declare integer courseNumber

Declare string courseTitle

Declare vector <string> prerequistes

WHILE EOF is not reached

READ in courseNumber

READ in courseTitle

READ in prerequisites if any exist

Store courseNumber, courseTitle, and any prerequisites in course object

Call Insert(course)

Function Insert(Course course)

IF root equals nullptr

root equals new Node(course)

ELSE

Call addNode(root, course)

Function addNode(Node\* node, Course course)

IF course number is less than node course number

IF left child equals nullptr

Left child equals new node

ELSE

Call addNode(node points to left, course)

ELSE

IF right child equals nullptr

Right child equals new node

ELSE

Call addNode(node points to right, course)

Print Course Information Pseudocode

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

Declare current node equal to root

WHILE current node does NOT equal nullptr

IF current node matches

OUTPUT course information

ELIF course number is less than current node course number

Set current node equal to left

ELSE

Set current node equal to right

**Menu**

Declare data structure object

Declare integer choice equal to 0

WHILE choice does NOT equal 4

OUTPUT Menu

OUTPUT 1. Load courses

OUTPUT 2. Print course list

OUTPUT 3. Print course

OUTPUT 4. Exit

INPUT choice

SWITCH (choice)

CASE 1: Call loadCourses()

Break

Case 2: Call displayCourses()

Break

Case 3: Call printCourseInformation()

Break

Case 4: Exit

Break

Default: OUTPUT choice not recognized

**Print Courses in Alphanumeric Order**

Vector

Call selectionSort(courses)

FOR (i equals 0; i is less than courses size; i++)

Call displayCourses(courses[i])

Function selectionSort(courses)

Declare integer i equal to 0

Declare integer j equal to 0

Declare integer min equal to 0

FOR (i equals 0; i is less than courses size -1; i++)

SET min equal to i

FOR (j equals i +1; j is less than courses size; j++)

IF course number at j is less than min course number

Min equals j

Swap course at i with course at min

Function displayCourses(course)

OUTPUT course number

OUTPUT course title

Hash Table

Call PrintAll()

Function PrintAll()

FOR (Iter equals begin; Iter does NOT equal end; iter++)

IF(key does NOT equal UINT\_MAX)

OUTPUT course number

OUTPUT course title

SET node equal to next iter

WHILE node NOT equal to nullptr

OUTPUT course number

OUTPUT course title

SET node equal to next node

Binary Search Tree

Call inOrder(root)

Function inOrder(Node\* node)

IF (node does NOT equal nullptr)

Call inOrder(node->left)

OUTPUT course number

OUTPUT course title

Call inOrder(node->right)

**Runtime Analysis**

| **Vector Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| Declare new Course object | 1 | 1 | 1 |
| Declare integer courseNumber | 1 | 1 | 1 |
| Declare string courseTitle | 1 | 1 | 1 |
| Declare vector <string> prerequistes | 1 | 1 | 1 |
| WHILE EOF is not reached  READ in courseNumber  READ in courseTitle  READ in prerequisites if any exist | 4 | n | 4n |
| Store courseNumber, courseTitle, and any prerequisites in course object | 1 | n | n |
| PUSH course object onto courses vector | 1 | n | n |
| **Total Cost** | | | 6n + 4 |
| **Runtime** | | | O(N) |

| **Hash Table** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| Declare new Hash Table object | 1 | 1 | 1 |
| Declare new Course object | 1 | 1 | 1 |
| Declare integer courseNumber | 1 | 1 | 1 |
| Declare string courseTitle | 1 | 1 | 1 |
| Declare vector <string> prerequistes | 1 | 1 | 1 |
| WHILE EOF is not reached  READ in courseNumber  READ in courseTitle  READ in prerequisites if any exist | 4 | n | 4n |
| Store courseNumber, courseTitle, and any prerequisites in course object | 1 | n | n |
| Insert Course object into Hash Table | 1 | n | n |
| **Total Cost** | | | 6n + 5 |
| **Runtime** | | | O(n) |

| **Binary Search Tree** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| Declare new Binary Search Tree object bst | 1 | 1 | 1 |
| Declare root equal to nullptr | 1 | 1 | 1 |
| Declare new Course object course | 1 | 1 | 1 |
| Declare integer courseNumber | 1 | 1 | 1 |
| Declare string courseTitle | 1 | 1 | 1 |
| Declare vector <string> prerequistes | 1 | 1 | 1 |
| WHILE EOF is not reached  READ in courseNumber  READ in courseTitle  READ in prerequisites if any exist | 4 | n | 4n |
| Store courseNumber, courseTitle, and any prerequisites in course object | 1 | n | n |
| Call Insert(course) | 1 | n | n |
| **Total Cost** | | | 6n+ 6 |
| **Runtime** | | | O(n) |

**Advantages and Disadvantages**

Vector

Advantages:

* The size of vectors are dynamic, therefore they can be increased.
* Vectors contain several member functions that allow for insertion and deletion.

Disadvantages:

* Searching for a particular element in a vector is O(n) due to potentially having to iterate through the entire vector.
* Insertion and deletion in the middle of a vector is very insufficient due to having to reallocate data.

Hash Table

Advantages:

* Hash tables are constant when searching, inserting, and deleting a particular element.

Disadvantages:

* Collisions can cause hash tables to become inefficient.

Binary Tree

Advantages:

* Binary trees can be implemented to maintain order after insertion so it’s easy to maintain a sorted list of the elements.
* Binary trees are O(log n) when searching, inserting, and deleting elements.

Disadvantages:

* Binary trees are more complex and require more memory.

**Recommendation**

Based on the Big O analysis and my analysis of each data structure, I would recommend using the hash table for this assignment. I recommend using a hash table, because hash tables are efficient for searching, inserting, and deleting elements. Each of these operations’ runtime is constant due to being able to hash a key value and locate a particular bucket in the hash table.