**Project One**

Arica Bryant

Southern New Hampshire University

CS-300 DSA: Analysis and Design

Professor David Ostrowski

**Pseudocode**

**File Input and Course Object Pseudocode for Vector**

CREATE empty courseInfo vector

SET input equal to TRUE

OPEN FILE

IF file opening is successful

LOOP through till end of file

READ current line

SPLIT current line at “,”

IF current line has less than 2 elements

THEN SET input equal to FALSE

BREAK from loop

ELSE

CREATE a new Course object

SET first section of current line equal to courseNumber

SET second section of current line equal to name

IF there are any prerequisties

SET remaining sections of current line equal to prerequisites

ADD the new Course object to the courseInfo vector

ELSE

DISPLAY error opening file message

CLOSE FILE

**Print Course Information Pseudocode for Vector**

FOR LOOPS through all courses

IF searched course is found

THEN output the course information

FOR each prerequisite of the course

PRINT the prerequisite course information

ELSE output “Course not found”

**File Input and Course Object Peudocode for Hash Table**

CREATE empty courseInfo vector

SET input equal to TRUE

OPEN FILE

IF file opening is successful

CREATE vector for data storage

CREATE HashTable function

LOOP through till end of file

READ current line

SPLIT current line at “,”

IF current line has less than 2 elements

THEN SET input equal to FALSE

BREAK from loop

ELSE

CREATE a new Course object

SET first section of current line equal to courseNumber

SET second section of current line equal to name

SET remaining sections of current line equal to prerequisites

SET courseNumber as keys

GET node using this key value

IF the node is not found

SET new node to key position

ELSE the node is found

IF the node has not been used

SET node key equal to new key value

SET node course equal to new course value

SET current pointer to null

ELSE there is a collision

WHILE LOOP until the next open node

THEN add new node to the end

ELSE

DISPLAY error opening file message

CLOSE FILE

**Print Course Information Pseudocode for Hash Table**

CREATE a key value

FOR LOOPS through all courses using key

IF the course is found for the key

THEN output the course information

FOR each prerequisite of the course

PRINT the prerequisite course information

ELSE output “Course not found”

**File Input and Course Object Pseudocode for Binary Tree**

CREATE Binary Search Tree function

SET root variable equal to null

OPEN FILE

IF file opening is successful

CREATE vector for data storage

LOOP through till end of file

READ current line

SPLIT current line at “,”

IF current line has less than 2 elements

THEN SET input equal to FALSE

BREAK from loop

ELSE

CREATE a new Course object

SET first section of current line equal to courseNumber

SET second section of current line equal to name

SET remaining sections of current line equal to prerequisites

INSERT Course object into Binary Search Tree using Insert() function

**Insert()**

IF current node is larger than the current courseNumber

IF there is no left node

THEN add a new node to the left

ELSE recurse down the left node

ELSE

IF there is no right node

THEN add a new node to the right

ELSE recure down the right node

ELSE

DISPLAY error opening file message

CLOSE FILE

**Print Course Information Pseudocode for Binary Tree**

FOR LOOPS through all courses using courseNumber

IF the course is found

THEN output the course information

FOR each prerequisite of the course

PRINT the prerequisite course information

ELSE output “Course not found”

**Menu Pseudocode**

CREATE Main() method

SET user choice to 0

PRINT “Menu:”

PRINT “1. Load Courses”

PRINT “2. Display Course List”

PRINT “3. Display Course”

PRINT “9. Exit Program”

WHILE user choice does not equal 9

IF user input equals 1

THEN load file data into a data structure

IF user input equals 2

THEN display all course information

IF user input equals 3

THEN find and print course specified by user

IF user input equals 9

THEN exit the loop

**Printing courses in alphanumeric order Pseudocode**

FOR all courses

IF node does equal null

InOrder not left

OUTPUT course number, course title

IF there are any prerequisites

OUTPUT preprequisites

InOrder right

**Big O Analysis and Evaluation**

**Vector**

|  |  |  |  |
| --- | --- | --- | --- |
| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| **Create empty vector** | 1 | 1 | 1 |
| **For each line in file** | 1 | n | n |
| **Create a Course object** | 1 | n | n |
| **SET first section equal to courseNumber** | 1 | n | n |
| **SET second section equal to name** | 1 | n | n |
| **IF there are any prerequisites** | 1 | n | n |
| **SET third section equal to prerequisites** | 1 | n | n |
| **Total Cost** | | | 6n + 1 |
| **Runtime** | | | O(n) |

**Hash Table**

|  |  |  |  |
| --- | --- | --- | --- |
| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| **Create empty vector** | 1 | 1 | 1 |
| **Create Hash Table** | 1 | 1 | 1 |
| **For each line in file** | 1 | n | n |
| **Create a Course object** | 1 | n | n |
| **Create a Key value** | 1 | n | n |
| **IF current node is not found** | 1 | n | n |
| **SET new node to key position** | 1 | n | n |
| **ELSE the node is found** | 1 | n | n |
| **IF the found node has not been used** | 1 | n | n |
| **SET node key equal to new key value** | 1 | n | n |
| **SET node course equal to new course value** | 1 | n | n |
| **SET the current pointer equal to null** | 1 | n | n |
| **ELSE there is a collision** | 1 | n | n |
| **WHILE LOOP until next open node** | 1 | n | n |
| **THEN add new node to the end** | 1 | n | n |
| **Total Cost** | | | 13n + 2 |
| **Run Time** | | | O(n) |

**Binary Search Tree**

|  |  |  |  |
| --- | --- | --- | --- |
| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| **Create Binary Search Tree function** | 1 | 1 | 1 |
| **For each line in file** | 1 | n | n |
| **Create Course object** | 1 | n | n |
| **SET first section equal to courseNumber** | 1 | n | n |
| **SET second section equal to name** | 1 | n | n |
| **IF there are any prerequisites** | 1 | n | n |
| **SET third section equal to prerequisites** | 1 | n | n |
| **IF root is null** | 1 | n | n |
| **IF current node is larger than current courseNumber** | 1 | n | n |
| **IF there is no left node** | 1 | n | n |
| **THEN add a new node to the left** | 1 | n | n |
| **ELSE recurse down the left node** | 1 | n | n |
| **IF there is no right node** | 1 | n | n |
| **THEN add a new node to the right** | 1 | n | n |
| **ELSE recurse down the right node** | 1 | n | n |
| **Total Cost** | | | 13n + 1 |
| **Run Time** | | | O(n) |

According to the total cost results of each data structure, the worst case running time for a vector is 6n+1, the worst case running time for a hash table is 13n+2, and the worst case running time for a Binary Search Tree is 13n+1. Comparatively, the Hash Table and Binary Search Tree have the worse costs while the vector has the best cost.

Vectors – Some advantages of using a vector are that elements can be accessed immediately which gives them a much quicker runtime especially when working with small amounts of data. A disadvantage is that when working with large sets of data, the program will take longer to sort through data.

Hash Tables – Hash Tables have the advantage of using key/value pairs. This can make finding and accessing the specific elements you need much easier since you can just search the hash for the key. A disadvantage is that you will need to take into account potential collisions when inserting/deleting data which can slow down runtime.

Binary Search Trees – An advantage of using Binary Search Trees for this project is that they have access to sorting functions such as InOrder, PreOrder, and PostOrder. An optimal function for this project would be the InOrder method because it will perform the necessary alphanumerical sorting on its own. However just like Hash Tables, because of their complexity, the runtime of a Binary Search tree may be larger and consume more memory.

Based on the BigO analysis and general advantages/disadvantages of the three data structures, I am choosing to use a Binary Search Tree for this project. While vectors are simple and easy to manage, their runtime is much slower when working with large sets of data. Hash Tables provide useful tools in the form of key/value pairs, but they require extra attention to ensure there are no collisions.