Ryne Williams

Southern New Hampshire University

CS 300 6-2 Project One

#Pseudocode to load page

Class CourseParser():

OpenFile

If (no file is found)

Print “File not found.”

Else

While (line count is not equal to 0)

If (file line parameters is less than 2)

Return error

Else

Read line parameters

If (parameter count is more than 2)

If (parameter is not on another line)

Print “Invalid prerequisite.”

Else

Read file line

Else

Print “Invalid line item.”

--file line

#Vector Data Structure Pseudocode

Class CourseInfo (Vector <course>, Courses) {

Vector <String> CourseNum

Vector <String> CourseName

Vector <String> Prerequisites

For (number of lines in file)

If (course info has 2 parameters)

CourseNum = first parameter

CourseName = second parameter

Else

CourseNum = first parameter

CourseName = second parameter

Prerequisites = remaining parameters

}

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

Accept input for course to print

If (course is in vector)

For (Course in CourseNum)

Print CourseNum, CourseName,

While (Prerequisite >0)

Print Prerequisite.

}

#HashTable Data Structure Pseudocode

number of Prerequisite Courses (Hashtable<Course> courses) {

set key to modulo(%) file size

create course key

get node from key

if node is equal to null pointer

create new node

insert course name data into name vector

insert course number data into number vector

for (number of prerequisites)

insert prerequisite into prereq vector

return course information

else

if temp node is equal to max key value

set key

set course info

set temp node pointer to null pointer

else

while temp node does not point to null pointer

point temp node to next

set next node equal to new node

}

void print Course Information(Hashtable<Course> courses, String courseNumber) {

set key to modulo(%) file size

create course key

get node from key

for all courses in hash table

if course node is the same as course number node

If key is not equal to maximum key value

Print course name, course number

For number of prerequisites

Print course prerequisites

}

#BinarySearchTree Pseudocode

int numPrerequisiteCourses(Tree<Course> courses) {

set tree’s root equal to null pointer

if root is equal to null pointer

insert course name into course name root node

insert course number into course num root node

for all prerequisites

insert prerequisite into prereq vector root node

else

//Create new node

if current node is larger than new node

if left child node is equal to null pointer

create new node at left child side

else

move current node down to left child node

else current node is smaller than new node

if right child node is equal to null pointer

create new node at right child side

else

move current node down to right child node

insert course name into course name new node

insert course number into course num new node

for all prerequisites

insert prerequisite into prereq vector new node

}

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

set current node equal to root node

if node does not equal null pointer

go to left child node

print course name

print course number

for all prerequisites

print course prerequisite

go to right child node

}

#Pseudocode to load data structures and print menu

Menu(): {

Call CourseParser class

LoadBids():

For all instances of course info in course file

Load course name into data structure

Load course number into data structure

For all prerequisites

Load prerequisites into data structure

While choice is not Exit

Print option 1: load course file

Print option 2: display all courses

Print option 3: display course name, number, and prerequisites

Print option 4: Exit

Case 1:

Call LoadBids()

Case 2:

Call printCourseInformation()

Print all courses

Case 3:

Call printCourseInformation()

Print course name, number, and prerequisites

Case 4:

Exit program

#Vector Sort Pseudocode

VectorSortAlphanumeric():

Create currentNode and point at second node in vector

Create previousNode and point at head node in vector

While currentNode is not equal to null pointer

If currentNode is less than previousNode

Swap currentNode and previousNode

Point currentNode at next node in vector

Else

Point currentNode at next node in vector

For all course information

Print course name, number, and prerequisites

#HashTable Sort Pseudocode

HashTableAlphanumeric():

Set first hash value equal to head node

Set tempNode equal to null

While tempNode is not equal to null pointer

if currentNode is less than tempNode

place tempNode at right

point tempNode to next node

else

place tempNode at left

for all course information

print course name, number, and prerequisites

#BinarySearchTree Pseudocode

BinarySearchTreeAlphanumeric():

Create new vector for course information

If node does not equal null pointer

If node has left child

Move node to left child

Else if node has right child

Place node data in vector

Else

Return node

Place node at next index in vector

Print course name, number, and prerequisites

Runtime Analysis:

| **Vector Sort** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **for all courses** | 1 | n | n |
| **if the course is the same as courseNumber** | 2 | n | n |
| **print out the course information** | 1 | 1 | 1 |
| **for each prerequisite of the course** | 1 | n | n |
| **print the prerequisite course information** | 1 | n | n |
| **Total Cost** | | | 5n + 1 |
| **Runtime** | | | O(n) |

| **Hash Table Sort** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **for all courses** | 1 | n | n |
| **if the course is the same as courseNumber** | 3 | n | n |
| **print out the course information** | 3 | 1 | 1 |
| **for each prerequisite of the course** | 1 | n | n |
| **print the prerequisite course information** | 1 | n | n |
| **Total Cost** | | | 8n + 1 |
| **Runtime** | | | O(n) |

| **Binary Search Tree Sort** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **for all courses** | 1 | n | n |
| **if the course is the same as courseNumber** | 3 | n | n |
| **print out the course information** | 2 | 1 | 1 |
| **for each prerequisite of the course** | 1 | n | n |
| **print the prerequisite course information** | 1 | n | n |
| **Total Cost** | | | 7n + 1 |
| **Runtime** | | | O(log n) |

Vector data structures are dynamic, meaning that their size can be changed easily and without modifying the source code. However, Vectors tend to take up more memory than other data structures and are best suited for when data is consistently changing. Hash tables use a key to determine where the data is stored in memory which can make finding such data much easier than most other data structures. However, hash tables tend to be prone to collision and are not easily sorted, due to the fact that it is stored based on the key, not the index. Binary search tree data structures are very efficient in searching and removing data, so long as the data is balanced in the tree. However, they are not very efficient when searching for random data in the binary search tree.

Given the advantages and disadvantages of each data structure, along with the runtime complexity of the coding, I feel as if a vector data structure would be the best option for this program. The vector data structure can be sorted to print all courses more efficiently than the other data structures. Also, it is more efficient and easy to change the courses in the vector if need be.