Emily Nagorski

UID: 1969938

CS-300-T6612

Project One

1. Resubmit pseudocode from previous pseudocode assignments and update as necessary.

Linked List Pseudocode

// define relevant course data information

class Course {

string courseNum

string courseName

vector preReq

}

// define how the program opens the file, reads the data from the file, parses each line, and checks for file format errors

vector string OpenFile (string file name) {

initialize vector string

initialize string to hold single line

initialize basic ifstream

open file name with ifstream

if file is not open (is\_open)

print error message to user

return file data

close file

return information from file

}

// show how to create course objects and store them in the appropriate data structure

vector <Course> CreateSchedule (vector <string>) {

initialize vector <Course>

initialize stringstream linestream

initialize string token, token will store each word from the line in document

initialize int count to track the number of tokens per line

for (each course)

set counter equal to 0

create new course

linestream, pull line content

pull token from linestream until the end of line

if (counter == 0)

course courseNumber = token

increase count, ++ count

else if (counter == 1)

courseName = token

increase count, ++count

else

if token = courseNumber that exists

push token to the end of course preReqs

else

print “Prerequisite is a previous course.”

increase count, ++count

if (counter <2)

print “File format error. Please review file for course name and number and try again.”

push course to end of courses

return courses

}

// search the data structure for a specific course and print out course information and prerequisites

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

for all courses

if the course is the same as courseNumber

print out the course information

for each prerequisite of the specified course

print the prerequisite course information

else course returned is empty

print “Course data not found”

return

}

Hash Table Pseudocode

// define relevant course data information

class Course {

string courseNum

string courseName

Vector <string> preReq

}

// define how the program opens the file, reads the data from the file, parses each line, and checks for file format errors

vector string OpenFile (string file name) {

initialize vector string

initialize string to hold single line

initialize basic ifstream

open file name with ifstream

if file is not open (is\_open)

print error message to user

return file data

close file

return information from file

}

// show how to create course objects and store them in the appropriate data structure

create default constructor with parameters

initialize variable for courses

open file to read

while file is open

store course objects in a hash table

logic to free storage when class is destroyed

calculate a hash value and return key table size

// search for courses

create key for given class

for key, course in course.items():

print (course.Id, course.name)

for prerequisite in course.prerequisites:

print(prerequisite)

while node is equal to null

if the current node matched

return node

set node equal to next

// search the data structure for a specific course and print out course information and prerequisites

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

for all courses

if the course is the same as courseNumber

print out the course information

for each prerequisite of the specified course

print the prerequisite course information

else course returned is empty

print “Course data not found”

return

}

Binary Tree Pseudocode

// define relevant course data information

class course {

string courseID

string courseName

vector<string> prerequisites

}

// define how the program opens the file, reads the data from the file, parses each line, and checks for file format errors

vector string OpenFile (string file name) {

initialize vector string

initialize string to hold single line

initialize basic ifstream

open file name with ifstream

if file is not open (is\_open)

print error message to user

return file data

close file

return information from file

}

// show how to create course objects and store them in the appropriate data structure

Course (constructor)

courseId = “”

courseTitle = “”

prerequisiteCount = 0

prerequisiteList = []

Structural node

Create left pointer

Create right pointer

Node

set left = null

set right = null

Node (course myCourse)

course = myCourse

return

Class binarySearchTree

Private

create node pointer, root

addNode(node, course)

Public

printAllCourses

PrintCourse(Nose \*node, string id)

validatePrerequisite(string)

insertCourse(course)

InsertCours (course, currentCourse)

If root = null

assign root = new Node(currentCourse)

Else

call addNode (root, currentCourse)

AddNode (Node \*node, course)

if node courseId > current courseId

if left node = null

set left node = new node

else

call addNode (left node, course)

else

if right node = null

set right node = new node

else

call addNode (right node, course)

PrintCourse (string)

if root getCourseId = current id

print course ID, title, number of prerequisites, name of prerequisites

else

printCourse (root, id)

1. Create pseudocode for a menu.

Menu

While user has not selected menu option 4

Print: “Main Menu”

“1. Load data structure”

“2. Print course list”

“3. Print course”

“4. Exit program”

“Please enter your desired action: ”

Switch (choice)

case 1:

call loadFile function (dataStructure, filename)

//all of the define how program opens file in the top three pseudocode sections

break

case 2:

Call printCourseList function (dataStructure)

break

case 3:

Print: “Enter desired course number: ”

course >> userInput

call findCourse fuction (dataStructure, course)

break

case 4:

Print: “goodbye!”

break

1. Design pseudocode that will print out the list of the courses in the Computer Science program in alphanumeric order.

While current node courseNum != null

If node > current node

move in front of current node

Else node <current node

node should remain behind current node

PrintCourses (vector<Node> nodes)

While currentNode.next != null

print currentNode course information

currentNode -> currentNode.next

Break

1. Evaluate the run-time and memory of data structures that could be used to address the requirements.

**Linked List**

|  |  |  |  |
| --- | --- | --- | --- |
| Line Description | Line Cost | Times Executed | Total Cost |
| Loading data | 1 | n | n |
| Searching courses | n | n | 1 |
| Printing/sorting | 2 | n | n |

**Hash Table**

|  |  |  |  |
| --- | --- | --- | --- |
| Line Description | Line Cost | Times Executed | Total Cost |
| Loading data | 2 | n | n |
| Searching courses | 1 | n | 1 |
| Printing/sorting | 4 | n | n |

**Binary Tree**

|  |  |  |  |
| --- | --- | --- | --- |
| Line Description | Line Cost | Times Executed | Total Cost |
| Loading data | 1 | n | n |
| Searching courses | 2 | n | 1 |
| Printing/sorting | 4 | n | n |

1. Explain the advantages and disadvantages of each structure in your evaluation.

Linked lists can be great for the implementation of data structures. They are easy to navigate when it comes to deleting and inserting nodes, and they use dynamic memory allocation, so only the memory that is needed is reserved. The flip side to dynamic memory allocation is that the pointed for a linked list also require additional memory storage, and the nodes are always accessed sequentially which is a slower method that some of the others we focused on. In contrast to a linked list, hash tables are much more efficient, as they can quick access key-value pairs, due to the way they are stored. Inserting, deleting, and searching hash tables typically can be done on O(1) time, and are frequently used in search engines because of their efficiency. The main downside of hash tables is that they can quickly become an expensive product. It takes a lot of time to implement a hash table that is of high quality, and if the time is not put in, hash tables have the habit of producing duplicate keys, and creating collisions. Binary trees utilize a hierarchical data structure where a root node has a left and right node, and they branch out from there. Like the other data structures, it is possible to insert, delete and search the tree. Each of these operations take O(log(n)) to be executed. I like a binary tree structure because they have the capability to show relationships between data, however deleting nodes is much more complex, and if a tree gets too large insertion/deletion/searching can be slow.

1. Make a recommendation for which data structure you will plan to use in your code.

I believe that based on the requirements from ABCU, how they want to access and view their data, and the amount of data they have, the hash table would be the best option for the client. The main feature it sounds like they would rely on is searching, and hash tables search the quickest, and deletion is simple. Linked lists are slow for searching and binary trees can get slow with too much data, therefore for their needs and cost analysis the hash table would be the best data structure to implement.