**CS 300 Pseudocode Document**

**Example Function Signatures**

Below is an example of a function signature that you can use as a guide to help address the program requirements using each data structure for the milestones. The pseudocode for finding and printing course information is also given below and depicted in bold to help you get started. The provided pseudocode is for a vector data structure, so you may use this pseudocode in your first milestone as is. The hash table and tree structures are also shown below. But these structures are left for you to do in future milestones.

//Vector - Milestone 1

void searchCourse(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 course**

**print the prerequisite course information**

}

void openFile(){

openFile, open name of file

check openFile name matches file name

if true,

read line by line

if data is not a string

print error message "Data must be presented in string format, no integers"

print "closing file, please edit to strings, and try again"

close file

break

print "finished reading file"

close file

else

print error message

}

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

openFile, open name of file

check openFile name matches file name

if true,

read line by line

if data is not a string

print error message "Data must be presented in string format, no integers"

print "Closing file, please edit data into strings, and try again"

close file

break

print "finished reading file"

else

print error message

close file

print "please enter course information in the following format, all course information MUST be separated by a comma: Course Number, Course Name, Prerequisites (if applicable)

intake information as a string

place data at beginning of vector

//vector sorting algorithm to place in the accurate spot

courses.sort()

write to csv file

close file

}

void alphaNum(Vector<Courses> courses){

while vector != nullptr

return vector

}

//Hash Table - Milestone 2

void searchCourse(HashTable<Course> courses, String courseNumber) {

for all courses

if course number = course number

print out course information

}

void openFile(){

open file, open file name

check file name matches file name

if true

read line by line

if data is not a string

print error message "Data must be presented in a string format, no integers"

print "Closing file, please edit data into strings, and try again"

Close file

break

print "Finished reading the file"

close file

else

print error message

}

void createCourse(HashTable<Course> courses, String courseNumber){

open file, open file name

check file name matches file name

if true

read line by line

if data is not a string

print error message "Data must be presented in a string format, no integers"

print "Closing file, please edit data into string, and try again"

Close file

break

print "Finished reading the file"

else

print error message

print "please enter course information in the following format, all course information MUST be separated by a comma: Course Number, Course Name, Prerequisites (if applicable)

intake information as a string

place data at beginning of vector

//Hash Table sorting algorithm to place data in the //accurate spot

hash to a spot using division of array size

write to csv file

close file

}

void alphaNum(HashTable<Course> courses){

while bucketPointer != nullptr

return bucket

}

//Binary Search Tree – Milestone 3

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

while string courseNumber doesn't equal key

if courseNumber < key

go left

else if course number > key

go right

return course number

}

void createCourse(Tree<Course> courses, string courseNumber){

while string courseNumber doesn't equal key

if courseNumber < key

go left

if coursenumber > coursenumber's right node, and < courseNumbers left node

insert new node with key's value

else if courseNumber > key

go right

if coursenumber

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

for all courses

if courseNumber == courseNumber

print courseNumber

else

print "Course not found"

}

void createCourse(Tree<Course> courses, string courseNumber){

open file, open file name

check file name matches file name

if true

read line by line

if data is not a string

print error message "Data must be presented in a string format, no integers"

print "Closing file, please edit data into string, and try again"

Close file

break

print "Finished reading the file"

else

print error message

courseNumber root equals CSCI300

intake string courseNumber

if new key < CSCI300 or new key > CSCI300

//go left

if new courseKey < root (300) and < old key

insert new node

else if new courseKey < root and > old key

push old key left insert new node

//go right

else if new courseKey > root and < right key

push old key right insert new node

else if new courseKey > root and > old key

insert new node

}

void openFile(Tree<Course> courses, string courseNumber){

open file, open file name

check file name matches file name

if true

read line by line

if data is not a string

print error message "Data must be presented in a string format, no integers"

print "Closing file, please edit data into strings, and try again"

Close file

break

print "Finished reading the file"

close file

else

print error message

}

void alphaNum(Tree<Course> courses, string courseNumber){

//go all way left

while pointer doesn't == nullptr

print courseNumber

}

//MENU

void menu(){

print "Please select from the following menu"

print "Option 1: Load the file data into the data structure"

print "Option 2: Print an alphanumerically ordered list of all the courses in the Computer Science department"

print "Option 3: Print the course title and the prerequisites for any individual course"

print "Option 9: Exit the program"

intake userval

if userval == 1

openFile()

if userval == 2

alphaNum()

if userval == 3

searchCourse()

if userval == 9

print "Exiting the program"

else

print "Enter a valid option"

}

**Example Runtime Analysis**

When you are ready to analyze the runtime for the Project One data structures for which you created the pseudocode, use the example chart below to support your work. This particular example is for printing course information when using the vector data structure. As a reminder, this is the same pairing that was bolded in the pseudocode from the first part of this document. The example only covers the search function for the vector structure. You do not have to complete your runtime analysis until Project One. However, working on your analysis now may help you understand the changes as you complete the milestones. Don’t forget to include your charts in Project One. You will submit Project One in Module Six.

|  |  |  |  |
| --- | --- | --- | --- |
| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| **for all courses** | 1 | n | n |
| **if the course is the same as courseNumber** | 1 | n | n |
| **for each prerequisite of the course** | 1 | 1 | 1 |
| **for each prerequisite of the course** | 1 | n | n |
| **print the prerequisite course information** | 1 | n | n |
| **Total Cost** | | | 4n + 1 |
| **Runtime** | | | O(n) |

create course vector model

|  |  |  |  |
| --- | --- | --- | --- |
| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| **openFile open name of file** | 1 | n | n |
| **check openFile name = file name** | 1 | n | n |
| **if true** | 1 | n | n |
| **read line by line** | 1 | n | n |
| **if data is not a string \_\_\_**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_print error message "Data must be presented in string format, no integers" \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_print "Closing file, please edit data into string, and try again" \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_close file \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_break \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_print "finished reading file" \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_else \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_print error message \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_close file \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_print "please enter course information in the following format....." \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_intake information as a string \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_place data at the beginning of vector \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_courses.sort() \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_write to csv file \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_close file | 1  1  1  1  1  1  1  1  1  1  1  1  1  1  1 | n  n  n  n  n  n  n  n  n  n  n  n  1  n  n | n  n  n  n  n  n  n  n  n  n  n  n  1  n  n |
| **Total Cost** | | | 18n + 1 |
| **Runtime** | | | O(n) |

alphaNum vector method

|  |  |  |  |
| --- | --- | --- | --- |
| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| **while vector != nullptr** | 1 | n | n |
| **return vector** | 1 | n | n |
| **Total Cost** | | | 2n |
| **Runtime** | | | 2(n) |

hash table search method

|  |  |  |  |
| --- | --- | --- | --- |
| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| **for all courses** | 1 | n | n |
| **if course number = course number** | 1 | n | n |
| **print course information** | 1 | n | n |
| **Total Cost** | | | 3n |
| **Runtime** | | | O(n) |

hash table create course method

|  |  |  |  |
| --- | --- | --- | --- |
| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| **open file, open file name** | 1 | n | n |
| **check file name matches file name** | 1 | n | n |
| **if true** | 1 | n | n |
| **read line by line** | 1 | n | n |
| **if data is not a string \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_print error message "Data must be presented... \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_print "Closing file, please edit data... \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_close file \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_break \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_print "Finished reading the file" \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_else \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_print error message \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ print "please enter course information in the following..\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_intake information as a string \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_place data at beginning of vector \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_hash to a spot using division of array size \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_write to csv file \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_close file** | 1  1  1  1  1  1  1  1  1  1  1  1  1  1 | n  n  n  n  n  n  n  n  n  1  1  1  n  n | n |
| **Total Cost** | | | 15n + 3 |
| **Runtime** | | | O(n) |

alpha num hash table method

|  |  |  |  |
| --- | --- | --- | --- |
| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| **while bucketPointer != nullptr** | 1 | n | n |
| **return bucket** | 1 | n | n |
| **Total Cost** | | | 2n |
| **Runtime** | | | 2(n) |

binary tree search method

|  |  |  |  |
| --- | --- | --- | --- |
| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| **for all courses** | 1 | n | n |
| **if courseNumber == courseNumber** | 1 | n | n |
| **print courseNumber** | 1 | 1 | 1 |
| **else** | 1 | n | n |
| **print "Course not found"** | 1 | n | n |
| **Total Cost** | | | 4n + 1 |
| **Runtime** | | | O(n) |

binary tree create course method

|  |  |  |  |
| --- | --- | --- | --- |
| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| **open file, open file name** | 1 | n | n |
| **check file name matches file name** | 1 | n | n |
| **if true** | 1 | n | n |
| **read line by line** | 1 | n | n |
| **if data is not a string \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_print error message "Data must be presented in a string... \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_print "Closing file, please edit data into string, and try again" \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_close file \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_break \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_print "Finished reading the file \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_else \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_print error message \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_courseNumber root = CSCI300 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_intake string courseNumber \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_if new key < CSCI300 or new key > CSCI300 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_if new courseKey < root and < old key \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_insert new node \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_else if courseKey < root and > old key \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_push old key left insert new node \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_else if new courseKey > root and < right key \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_push old key right insert new node \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_else if new course key > root and > old key \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_insert new node** | 1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1 | n  1  1  n  n  1  n  1  n  n  n  n  n  n  n  n  n  n  n | n  1  1  n  1  n  1  n  n  n  n  n  n  n  n  n  n  n |
| **Total Cost** | | | 18n + 4 |
| **Runtime** | | | O(n) |

|  |  |  |  |
| --- | --- | --- | --- |
| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| **go all way left** | 1 | n | n |
| **while pointer != nullptr** | 1 | n | n |
| **print courseNumber** | 1 | 1 | 1 |
| **Total Cost** | | | 2n + 1 |
| **Runtime** | | | O(n) |

all way left

while pointer doesn't == nullptr

print courseNumber

Advantages and Disadvantages

Vectors, hash tables, and data trees have many advantages to storing data. Vectors can hold data in a straight line, easily add data to anywhere in the array. Hash tables can store lots of data in a single bucket. Data trees are easily searchable, and are commonly used in file storage systems. However each one has some disadvantages. While parsing through a vector, you need to start your search from the beginning or the end to find the data you are searching for. Worst case scenario you have to search the entire list to find out that the information you are seeking isn't there to begin with. Hash tables are easily searchable, by using the key to look up each item. However, if two or more items are assigned the same bucket, you would encounter a collision and thus make searching more time consuming. In the case of a collision, either a new bucket needs to be found, or you need to add more buckets to your list of buckets to store more information, unless you chain the buckets to add more data, to a signle bucket. Data trees are easy to search for data, you start at the root, and go either left if the data you are searching for is less than the root, or right if the data is greater than the root. The setback that you might encounter with a data tree is its limitations in that each tree branch can only have a maximum of two leafs, which limit how much data can be stored, along with maximizing storage and minimizing searching functions.

Reccomendations

With the data presented I would reccomend the hash table. Even though, adding to the bucket size is a tedious process for the computer to do, when the user needs to add to the list, if chaining is utilized correctly, the list should only have to hold ten buckets, as the storage system will hold courses 100, 110, 120, 130, 140, 150, 160, 170, 180, 190, each bucket can hold N items, and any course that has ends in the ten's place will go into its own bucket, minimizing collisions, and when searching for a specific course, makes it go quickly.