6-2 Submit Project One

Pseudocode

1. Resubmit pseudocode from previous pseudocode assignments and update as necessary. In the previous assignments, you created pseudocode for each of the three data structures (vector, hash table, and tree). Be sure to resubmit the following pseudocode for each data structure.
   1. Design pseudocode to define how the program opens the file, reads the data from the file, parses each line, and checks for formatting errors.
   2. Your pseudocode should show how to create course objects, so that one course object holds data from a single line from the input file.
   3. Design pseudocode that will print out course information and prerequisites.

**3-3 Milestone: Vector Data Structure Pseudocode**

function loadData(vector<>)

openfile(courseInformation)

readfile(courseInformation)

function storeData(vector<>)

for each data

store value in vector item

function printData(vector<>)

for each vector item

print vector item

function openfile(courseinformation)

Open file courseInformation

if error opening

print "error opening"

function readfile(courseinformation)

for each line from the file

if not (each line has at least 2 parameters ) OR (line ends with a prerequisite and there is no starting line with the same code)// (the two conditions to be checked here)

print "error found"

else

storedata(vector)

class course

private

string courseID

string courseName

vector<string> prerequisite

public

function storeDataCourse(){

loadData(vector<>)

storeData(vector<>)

}

function 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 course**

**print the prerequisite course information**

}

**4-3 Milestone: Hash Table Data Structure Pseudocode**

function loadData(vector<>)

openfile(courseInformation)

readfile(courseInformation)

function storeData(vector<>)

for each data

extract key, value from data

storeHashTable(key,value)

function printData(vector<>)

for each vector item

print vector item

function openfile(courseinformation)

Open file courseInformation

if error opening

print "error opening"

function readfile(courseinformation)

for each line from the file

if not (each line has at least 2 parameters ) OR (line ends with a prerequisite and there is no starting line with the same code)// (the two conditions to be checked here)

print "error found"

else

storedata(vector)

class course

private

string courseID

string courseName

vector<string> prerequisite

public

function storeDataCourse(){

loadData(vector<>)

storeData(vector<>)

}

function storeHashTable(key,value){

hashValue = hashFunction(key)

if hashTable[HashValue] is empty

insert key,value in hashTable[hashValue]

else

increment hashValue until hashTable[hashValue] is empty

insert key,value in hashTable[hashValue]

}

function hashFunction(key){

return key%tableSize

}

function 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 course

print the prerequisite course information

}

**5-3 Milestone: Tree Data Structure Pseudocode**

function loadData(vector<>) {

openfile(courseInformation)

readfile(courseInformation)

}

function storeData(vector<>){

for each data

extract key, value from data

storeHashTable(key,value)

}

function printData(vector<>){

for each vector item

print vector item

}

function openfile(courseinformation){

Open file courseInformation

if error opening

print "error opening"

}

function readfile(courseinformation) {

for each line from the file

if not (each line has at least 2 parameters ) OR (line ends with a prerequisite and there is no starting line with the same code)// (the two conditions to be checked here)

print "error found"

else

storedata(vector)

}

function createCourseObject(key, value) {

// Create a new course object

course = new Course()

// Parse the key and value to extract course information

courseID = extractCourseID(key)

courseName = extractCourseName(key)

prerequisites = extractPrerequisites(value)

// Set course attributes

course.setCourseID(courseID)

course.setCourseName(courseName)

course.setPrerequisites(prerequisites)

// Insert course object into the tree data structure

tree.insert(courseID, course)

}

class course

private:

string courseID

string courseName

vector<string> prerequisite

public:

// Setter methods

function setCourseID(courseID) { this.courseID = courseID }

function setCourseName(courseName) { this.courseName = courseName }

function setPrerequisites(prerequisites) { this.prerequisites = prerequisites }

}

function printCourseInformation(Tree courses, String courseNumber) {

course = tree.search(courseNumber)

if course is not null

print out the course information

for each prerequisite of the course

print the prerequisite course information

else

print "Course not found"

}

Create pseudocode for a menu. The menu will need to perform the following:

* 1. Load Data Structure: Load the file data into the data structure. Note that before you can print the course information or the sorted list of courses, you must load the data into the data structure.
  2. Print Course List: This will print an alphanumerically ordered list of all the courses in the Computer Science department.
  3. Print Course: This will print the course title and the prerequisites for any individual course.
  4. Exit: This will exit you out of the program.

Create pseudocode for a menu

int userChoice;

Output << “Enter choice: “<< endl;

Output << “1. Load data.” << endl;

Output << “2. Display Courses.” << endl;

Output << “3. Display Course.” << endl;

Output << “4. Exit.” << endl;

While (userChoice != 4)

Switch (choice)

Case 1: loadData();

Break;

Case 2: printCourseList();

Break;

Case 3: printCourse();

Break;

1. Design pseudocode that will print out the list of the courses in the Computer Science program in alphanumeric order.Continue working with the Pseudocode Document linked in the Supporting Materials section. Note that you will be designing for the same three data structures that you have been using in your previous pseudocode milestones (vector, hash table, and tree). This time you will create the final pieces of pseudocode that you will need for ABCU’s advising program. To complete this part of the process, do the following:
   1. Sort the course information by alphanumeric course number from lowest to highest.
   2. Print the sorted list to a display.

VECTOR

function printCourseList(Vector courses)

sortedVector = sort(vector)

for course in sortedVector:

Print display course

HASH

function printCourseListHashTable()

courseList = hashTable.values()

sortedCourseList = sort(courseList)

for course in sortedCourseList:

Print display course

TREE

functon printCourseListTree():

sortedCourses = tree.inOrder()

for course in sortedCourses:

Print display course

Evaluate the run-time and memory of data structures that could be used to address the requirements. In a previous assignment, you created pseudocode to do the following:

* 1. Define how the program opens the file, reads the data from the file, parses each line, and checks for formatting errors.
  2. Show how to create course objects, so that one course object holds data from a single line from the input file.
  3. Using this pseudocode written for the previous assignments, analyze the worst-case running time of each, reading the file and creating course objects, which will be the Big O value. This should not include the pseudocode written for the menu or the sample schedule above. To do this, do the following:Specify the cost per line of code and the number of times the line will execute. Assume there are n courses stored in the data structure.
  4. Assume the cost for a line to execute is 1 unless it is calling a function, in which case the cost will be the running time of that function.

1. Based on the advisor’s requirements, analyze each data structure (vector, hash table, and tree). Explain the advantages and disadvantages of each structure in your evaluation.
2. Now that you have analyzed all three data structures, make a recommendation for which data structure you will plan to use in your code. Provide justification for your recommendation, based on the Big O analysis results and your analysis of the three data structures.

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

| **Hash Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **for all courses** | 2 | n | n |
| **if the course is the same as courseNumber** | 1 | n | n |
| **print out the course information** | 1 | 1 | 1 |
| **for each prerequisite of the course** | 2 | n | n |
| **print the prerequisite course information** | 4 | n | n |
| **Total Cost** | 9n + 1 | | |
| **Runtime** | O(n) | | |

| **Tree Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **for all courses** | 1 | n | n |
| **if the course is the same as courseNumber** | 1 | n | n |
| **print out the course information** | 2 | 1 | 1 |
| **for each prerequisite of the course** | 1 | n | n |
| **print the prerequisite course information** | 4 | n | n |
| **Total Cost** | 7n + 1 | | |
| **Runtime** | O(n) | | |

Advantages:

Vector- Items get sorted when they get inserted

Hash- Very fast access to elements which makes it easier to add and remove

Tree- Searching for items is very efficient

Disadvantages:

Vector- Not as good for adding and removing items

Hash- Information is not sorted in order

Tree- Require significant amounts of memory to store data

After analyzing all three data structures I would say that my recommendation would be to use a tree data structure. Although significant memory is needed in order to store data in a tree, I think it will be the most efficient for adding, removing and sorting the data then the other data structures analyzed.