Project One Pseudocode: Vector Database

Bool FileChecker(file){

Open file

string file

string coursecheck

HashTable courseNumber

if file is open

while(getline(courseFile, file)){

split line by comma;

add element one to course number hashtable;

add file to course hashtable

counter = 0

while loop through hashtable

if courseCheck has more than 2 elements

compare elements 3 on to course number hashtable;

if element is not in hashtable;

close file

end

update counter

Return course list

Close course file

Void createDatabase(hashtable, \*vector)

Create course object NewCourse

Get length of the hashtable

While counter is not equal to length of hashtable{

newCourse = hashtable element

add new course to vector

update counter

Void PrintCourses(\*vector){

Create counter

While counter is less than length of vector{

Print out vector at counter

Update counter

// Tree pseudocode

int numPrerequisiteCourses(Tree<Course> courses) {

totalprerequitsites = prerequisites of course

currentCourse = root

while currentCourse is not equal to null pointer

if root is equal to course

add course to total prerequisites

else

keep traversing tree

}

void printSampleSchedule(Tree<Course> courses) {

for course in courses

print course

}

void printCourseInformation(Tree<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 selectionMenu(){

Print out:

1. Load data structure
2. Print course List
3. Print course
4. Exit

}

## Example Runtime Analysis

When you are ready to begin analyzing the runtime for the data structures that you have created pseudocode for, use the chart below to support your work. This 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.

**RunTime**

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **If file is open** | 1 | 1 | 1 |
| **while(getline(courseFile, file)){** | 1 | n | n |
| **Split line by commas** | 1 | n | n |
| **add element one to course number hashtable;** | 1 | n | n |
| **add file to course hashtable** | 1 | n | n |
| **while loop through hashtable** | 1 | n | n |
| **if courseCheck has more than 2 elements** | 1 | n | n |
| **compare elements 3 on to course number hashtable;** | 1 | n | n |
| **if element is not in hashtable;** | 1 | n | n |
| **Total Cost** | | | 8n + 1 |
| **Runtime** | | | O(n) |

Evaluation

The way I wrote my pseudocode is that one function pulled all the information from the file a put it into a hashtable. I then had the tree and binary tree pull from that hashtable.

The hashtable has the advantage of being able to access a lot fast since there is a key associated with each of the things. The disadvantage is that it can take time to load the data into the table.

For a vector the advantage is that is easy to fill and is fast. The disadvantage is that to find the information that you need you will have to loop through the entire list. The makes the run time a lot longer.

For a Binary tree the advantage is that the information is stored in an orderly manner and the fetching of the information is relatively fast. The disadvantage is a bit like the vector you have to run through several other levels of the tree before you get to the one you would like. So the worst case run time would be equivalent to the height of the tree.

I realize now that might not be what was wanted, but I felt it was the best way to do it. This being said I feel that the hashtable is the most efficient way to go since the worst run time would be O(n)