6-2 Project One

Craig A Nelson

Department of Computer Science, Southern New Hampshire University

CS-300-T3671 DSA: Analysis and Design 22EW3

Professor David Ostrowski

February 13th, 2022

You will be writing pseudocode to address each of advising’s requirements, and you will do so for each of the data structures you already started to explore in the previous assignments (vector, hash table, and tree). Then you will perform a runtime analysis to determine which data structure will be the best to use when you begin coding in the next project.

**Pseudocode from Earlier Submissions.**

**// Vector pseudocode**

int numPrerequisiteCourses(Vector<Course> courses, Course c) {

totalPrerequisites = prerequisites of course c

for each prerequisite p in totalPrerequisites

add prerequisites of p to totalPrerequisites

print number of totalPrerequisites

}

void printSampleSchedule(Vector<Course> courses) {

for all courses

print course name

if course has prerequisites

for each prerequisite

print prerequisite

}

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 course

print the prerequisite course information

}

HashTable Notes

Time Complexity:

Search: O(1+(n/m))  
Delete: O(1+(n/m))  
  
where n = Number of slots in Hash table   
m = Number of keys to be inserted

Here n/m is the Load Factor.  
Load Factor (∝) must be as small as possible.

/ **Hashtable Pseudocode**

int numPrerequisiteCourses(Hashtable courses, Course c) {

totalPrerequisites = Hashtable[c]

for each prerequisite p in totalPrerequisites

add prerequisites in Hashtable[p] to totalPrerequisites

print number of totalPrerequisites

}

void printSampleSchedule(Hashtable courses) {

for all key, value pair in courses

print key course name

if value has prerequisites

for each prerequisites

print prerequisites

}

void printCourseInformation(Hashtable courses, String courseNumber) {

for all courses

if the course is the same as courseNumber

print out the course information

for each prerequisite of the Hashtable[course]

print the prerequisite course information

}

**// Tree pseudocode**

int numPrerequisiteCourses(Tree courses, Node c) {

totalPrerequisites = left and right child of Node c

for each prerequisite p in totalPrerequisites

add left and right Nodes of node p to totalPrerequisites

print number of totalPrerequisites

}

void printSampleSchedule(Tree courses) {

for all Nodes as courses

print course name

if course has left node

print left node as prerequisites

if course has right node

print right node as prerequisites

}

void printCourseInformation(Tree courses, String courseNumber) {

for all Nodes

if the course is the same as courseNumber

print out the node's information

if course has left node

print left node as prerequisite course information

if course has right node

print right node as prerequisite course information

end Function

else

if course has left node

goto left node

if course has right node

goto right node

}

Then you will perform a runtime analysis to determine which data structure will be the best to use when you begin coding in the next project.

**Runtime Analysis**

**Vector (given in the lesson)**

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 course

print the prerequisite course information

}

Table

Description automatically generated

**Hashtable**

void printCourseInformation(Hashtable courses, String courseNumber) {

for all courses

if the course is the same as courseNumber

print out the course information

for each prerequisite of the Hashtable[course]

print the prerequisite course information  
}

|  |  |  |  |
| --- | --- | --- | --- |
| **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 | 1 | 1 | 1 |
| for each prerequisite of the Hashtable[course] | 1 | n | n |
| print the prerequisite course information | 1 | n | n |
| **Total Cost** | | | **4n + 1** |
| **Runtime** | | | **O(n)\*** |

\* Hashtables usually have an O(1) runtime but in this scenario the hashtable must be iterated over the courses giving it a O(n) runtime.

**Binary Search Tree**

void printCourseInformation(Tree courses, String courseNumber) {

for all Nodes

if the course is the same as courseNumber

print out the node's information

if course has left node

print left node as prerequisite course information

if course has right node

print right node as prerequisite course information

end Function

else

if course has left node

goto left node

if course has right node

goto right node

}

|  |  |  |  |
| --- | --- | --- | --- |
| **Code** | **Line Cost** | **#Times**  **Executes** | **Total**  **Cost** |
| for all Nodes | 1 | n | n |
| if the course is the same as courseNumber | 1 | n | n |
| print out the node's information | 1 | 1 | 1 |
| if course has left node | 1 | n/2 | n/2 |
| print left node as prerequisite course information | 1 | n | n |
| if course has right node | 1 | n/2 | n/2 |
| print right node as prerequisite course information | 1 | 1 | 1 |
| **Total Cost** | | | **2(n/2) + 3n + 2** |
| **Runtime** | | | **O(n)\*** |

\* Binary Search Trees usually have an O(logn) but this program is having to traverse every node. Traversals are Big O notation of O(n). (Stack Overflow(n.d))

A picture containing diagram

Description automatically generated

*Picture from Educative. (n.d.).*

Design pseudocode that will print out course information and prerequisites.

Create pseudocode for a menu.   
Load Data Structure:  
Print Course List:  
Print Course:   
Exit:

**Menu Pseudocode**

print welcome message

first char string convert to int

while choice does not equal exit

print menu

print choose?

if choice = 1 //Load Data Structure

Output " Enter File Name”

If correct file entered

print “loaded successfully”

else incorrect file

print “no file found”

return menu

print choose?

if choice = 2 //Print Course List

print “sample”

print sample schedule()s //course IDs alphabetized

else

print “load classes first”

if choice = 3 //Print Course Info

//validate choice

print “enter courseID from list for info”

for userChoice(type) = coursed

print course information

else

print “load classes first”

if choice != 9

print “ invalid entry”

else

print “ TY for using course planner”

}

exit

**Advantages and Disadvantages**

**For a Hash table vs Binary Search Table**

**Table

Description automatically generated**

“If you know the size of the input, then you can use the Hash Table. But if you are not sure about the input size, then you should go with BST. Also, if you are not so familiar with the input size, but after inserting all the data, the operation that you are following is retrieval, deletion, then you should use the Hash Table. But if you are continuously inserting, updating, deleting, and retrieving data, then go for BST.” AfterAcademy. (n.d.).

From the table above we see the comparison between Hashtable and BST. Determining which to use would depend on the problem we are faced with.

**Conclusion**

By preparing all the pseudocodes and analyzing the runtimes. I have chosen to program with Binary Search Tree to do the final C++ Program. For such little data that we are using there should be no disadvantages or advantages of using any of the vector, hashtable, or BST. The code though could be used on a grander scale for a large university or college. I think BST would give the best outcome.

**References**

Baeldung. (2021, October 13). Binary trees vs. linked lists vs. hash tables. Baeldung on Computer Science. Retrieved February 16, 2022, from <https://www.baeldung.com/cs/binary-trees-vs-linked-lists-vs-hash-tables>

Big O notation: A primer for beginning devs. Educative. (n.d.). Retrieved February 16, 2022, from <https://www.educative.io/blog/a-big-o-primer-for-beginning-devs>

Binary search tree vs hash table. AfterAcademy. (n.d.). Retrieved February 16, 2022, from https://afteracademy.com/blog/binary-search-tree-vs-hash-table

Looking for clarification on hashing and BST functions and big O notation. Stack Overflow. Retrieved February 16, 2022, from https://stackoverflow.com/questions/13816713/looking-for-clarification-on-hashing-and-bst-functions-and-big-o-notation