**6-2 Project One**

Tyler Barnes

Southern New Hampshire University

CS 300 Analysis and Design

13 February 2024

**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)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 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 course is same as courseNumber

Print the course information

For each prerequisite of Hashtable[course]

Print prerequisite course info

}

**Tree pseudocode**

OpenFile()

Initialize BinaryTree

Initialize ifstream instream

Open file with file name

IF file opens

Continue

ELSE

Print “File did not open”

Exit

Check file

IF two parameters

IF int and string correct order

Continue parsing list

ELSE

Print “invalid parameters”

Exit

CourseCreation()

String Course

Int course number

Open file

WHILE open

IF node > course number

IF left node = nullptr

Left node = new node

ELSE

This add node(left node, course number)

ELSE

IF right node = nullptr

Right node = new node

ELSE

This add node(right node, course number)

Close file

Return to Menu

SearchingCourse()

Open file

WHILE open

Int courseNumSearch

Input courseNumSearch

courseNumSearch

current node = root

WHILE current node doesn’t have nullptr

IF current node = course number

Return current course

IF course id is < current course number

Traverse left subtree

ELSE

Traverse right subtree

Return bid

PrintCourse()

IF node != nullptr

inOrder left node

Output course info

inOrder right node

ELSE

Return

Menu

Print menu:

Add course

Delete Course

Check course information and prerequisites

Exit

MenuSelection

Selection = 0

IF option == 1

Cin Course Name

Cin Course Number

ELSE IF == 2

Cin course to delete

Parse file

IF Delete course = course name

Delete course

ELSE

Print this course currently not listed

ELSE IF == 3

Parse file

IF courseNumSearch == courseNum

Print course information

ELSE IF == 4

“Goodbye”

Exit the program

**Runtime Analysis**

**VECTOR**

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 course is same as courseNumber

Print the course information

For each prerequisite of Hashtable[course]

Print prerequisite course info

}

|  |  |  |  |
| --- | --- | --- | --- |
| **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)\*** |

**Binary Search Tree & Menu**

PrintCourse()

IF node != nullptr

inOrder left node

Output course info

inOrder right node

ELSE

Return

Menu

Print menu:

Add course

Delete Course

Check course information and prerequisites

Exit

MenuSelection

Selection = 0

IF option == 1

Cin Course Name

Cin Course Number

ELSE IF == 2

Cin course to delete

Parse file

IF Delete course = course name

Delete course

ELSE

Print this course currently not listed

ELSE IF == 3

Parse file

IF courseNumSearch == courseNum

Print course information

ELSE IF == 4

“Goodbye”

Exit the program

|  |  |  |  |
| --- | --- | --- | --- |
| **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)\*** |

**Conclusion**

Making the pseudocodes and analyzing the runtimes, I chose the program Binary Search Tree. There’s little data that we would use and should be no disadvantages or advantages using vector, hashtable, or binary search tree. I believe BST would have the best outcome.