Sergio H. Passos

5 August 21

CS 300

6-2 Project One

void printMenu() {

choice = 0

while choice != 9:

print 1. Load Data Structure

print 2. Print Course List

print 3. Print Course

print 9. Exit

print Enter Choice:

choice = input

switch choice:

1:

// Load Data Structure

loadData()

2:

// Print Course List

Loop data:

data.printCourseInformation()

3:

// input

Print “which course:”

course = input

// Print Course

course = data.findCourse(course)

print course.info()

9:

// exit

break

}

// file input

Use iostream

Use ifstream

Use string

Ifstream file(“file\_name.txt”)

If file.is\_open():

String line

While getline(file, line):

Print line

Else

Print “Error: file could not be opened”

// 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 schedule

}

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<Course> courses) {

numPrerequisiteCourses = 0

for courses:

if courses.prerequisite:

numPrerequisiteCourses++

return numPrerequisiteCourses

}

void printSampleSchedule(Hashtable<Course> courses) {

for key: courses:

print courses.get(key) schedule

}

void printCourseInformation(Hashtable<Course> courses, String courseNumber) {

for key in courses:

if courses.get(key).courseNumber == courseNumber:

print courses.get(key) course information

}

// Tree pseudocode

int numPrerequisiteCourses(Tree<Course> courses) {

if courses.root is null

return 0

else

count = 0

loop through courses.left:

if course-> has prerequisite

count++

loop through courses.right:

if course-> has prerequisite

count++

return count

}

void printSampleSchedule(Tree<Course> courses) {

loop courses.left

print course sample schedule

loop courses.right

print course sample schedule

}

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

if courses.root.courseNumber < courseNumber:

loop courses.left

if courseNumber == course->courseNumber:

print course information

else:

loop courses.right

if courseNumber == course->courseNumber:

print course information}

## 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.

| **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 course** | 1 | n | n |
| **print the prerequisite course information** | 1 | n | n |
| **Total Cost** | | | 4n + 1 |
| **Runtime** | | | O(n) |

// hash table

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| for key in courses: | 1 | n | n |
| if courses.get(key).courseNumber ==  courseNumber: | 1 | n | n |
| print courses.get(key) course  information | 1 | 1 | 1 |
| **Total Cost** | | | 2n + 1 |
| **Runtime** | | | O(n) |

// binary tree

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| IF courses.root.courseNumber < courseNumber: | 1 | 1 | 1 |
| loop courses.left | 1 | n | n |
| if courseNumber == course-  >courseNumber: | 1 | n | n |
| print course information | 1 | 1 | 1 |
| Else: | 1 | 1 | 1 |
| loop courses.right | 1 | n | n |
| if courseNumber == course-  >courseNumber: | 1 | n | n |
| print course information | 1 | 1 | 1 |
| **Total Cost** | | | 2n + 2 |
| **Runtime** | | | O(n) |

## Runtime Analysis

From the analysis of all three data structures, I would right now go with a has table. Its performances the best considering the other two data structures. A binary tree is more memory efficient since it only allocated the required space when needed. Comparatively, a vector data structure needs to allocate its size and resize if exceeding the maximum allotted elements. Since we don’t know the size of the input it would be best to use a binary tree data structure so that it can be scalable for the future. Additionally, the binary tree has a root node which can be the base of the search; since to the left of the node will always be lesser, and to the right will always be greater. Automatically cutting the searching in half by a single comparison.