CS 300 6-2 Project One

Leanna Walker 10/12/2025

Milestone One: Vector Pseudocode

Start

CREATE a list called allCourses

WHILE true

PRINT “1. Load Course Files”

PRINT “2. Print All Courses”

PRINT “3. Search for a Course”

PRINT “9. Exit”

PRINT “Please make a selection: ”

GET userChoice (input)

IF the user entered “9”

PRINT “Goodbye!”

EXIT program

END IF statement

IF the user entered “1”

Attempt to OPEN the file

IF the file fails to open

PRINT “Error: file not found”

EXIT program

WHILE there are still lines left to read in the file

READ the next line in the file

SPLIT the line into parts using the comma as a divider

STORE the parts in a list (parts list)

IF the number of parts in the parts list is less than 2 THEN

PRINT “Error: Invalid format. Every line must have a course number and a title.”

EXIT program

ELSE

ADD the course number to new list (trueCourseNumbers list)

CREATE a new object called courseDetails to store course information

SET courseDetails.courseNumber to parts[0]

SET courseDetails.courseTitle to parts[1]

CREATE a new list inside courseDetails called prerequisites

IF the number of parts in the parts list is more than 2

SET a counter i to 2

WHILE i is less than the total number of parts

ADD parts[i] to courseDetails.prerequisites

INCREASE i by 1

FOR each prereq in course.Details.prerequisites  
 IF prereq is not found in trueCourse.Numbers list

PRINT “Error: Prerequisite ” + prereq + “ for course ” + courseDetails.courseNumber + “ is not a valid course.”

EXIT program

ADD the courseDetails object to the allCourses list

PRINT “All courses have been loaded.”

END IF statement

IF the user entered “3”

PRINT “Please enter a course number: ”

GET searchTerm (input)

FOR each course in the allCourses list

IF course.courseNumber is equal to the searchTerm

PRINT course.courseNumber, course.courseTitle  
 PRINT “Prerequisites:”  
 FOR EACH prereq in course.prerequisites  
 PRINT prereq

EXIT loop

IF the course was not found in the list

PRINT “ERROR: Course was not found.”

END IF statement

IF the user entered “2”:

PRINT “Sorted course list:”

SORT the allCourses list by courseNumber

FOR every course in allCourses

PRINT course.courseNumber, course.courseTitle

END IF statement

Milestone Two: Hash Table Pseudocode

Start

CREATE a hash table named courseHashTable  
CREATE a list named courseList

WHILE true

PRINT “1. Load Course Files”

PRINT “2. Print All Courses”

PRINT “3. Search for a Course”

PRINT “9. Exit”

PRINT “Please make a selection: ”

GET userChoice (input)

IF the user entered “9”

PRINT “Goodbye!”

EXIT program

END IF statement

IF the user entered “1”

Attempt to OPEN the file   
IF the file fails to open

PRINT “Error: file not found.”

EXIT program

WHILE there are still lines left to read in the file

READ the next line in the file

SPLIT the line into parts using the comma as a divider

STORE the parts in a list (parts list)

IF the number of parts in the parts list is less than 2 THEN

PRINT “Error: Invalid format. Every line must have a course number and a title.”

EXIT program

CREATE a new object called courseDetails to store course information

SET courseDetails.courseNumber to parts[0]

SET courseDetails.courseTitle to parts[1]

CREATE a new list inside courseDetails called prerequisites

IF the number of parts in the parts list is more than 2

SET a counter i to 2

WHILE i is less than the total number of parts

ADD parts[i] to courseDetails.prerequisites

INCREASE i by 1

STORE courseDetails in courseHashTable WITH KEY courseDetails.courseNumber

ADD courseDetails to the courseList

FOR each currentcourse in courseList

FOR each prereq in currentCourse.prerequisites

FIND lookupResult in courseHashTable WITH KEY prereq

IF lookupResult is not found THEN

PRINT “Error: Prerequisite ” + prereq + “ for course ” + currentCourse.courseNumber + “ is not a valid course.”

EXIT program

PRINT “All courses have been loaded.”

END IF statement

IF the user entered “3”

PRINT “Please enter a course number: ”

GET searchTerm (input)

FIND course in courseHashTable WITH KEY searchTerm

IF a course is found THEN

PRINT course.courseNumber, course.courseTitle

PRINT “Prerequisites: ”

FOR EACH prereq in course.prerequisites

PRINT prereq

ELSE

PRINT “Sorry, course was not found.”

END search

IF the user entered “2”

PRINT “Sorted course list: “

SORT the courseList by courseNumber

FOR every course in courseList

PRINT course.courseNumber, course.courseTitle

END IF statement

Milestone Three: Binary Search Tree

Start

CREATE a binary search tree named abcTree

WHILE true

PRINT “1. Load Course Files”

PRINT “2. Print All Courses”

PRINT “3. Search for a Course”

PRINT “9. Exit”

PRINT “Please make a selection:”

GET userChoice (input)

IF the user entered “9”

PRINT “Goodbye!”

EXIT program

IF the user entered “1”

Attempt to OPEN the file

IF the file fails to open

PRINT “Error: file not found.”

EXIT program

WHILE there are still lines left to read in the file

READ the next line in the file

SPLIT the line into parts using the comma as a divider

STORE the parts in a list (parts list)

IF the number of parts in the parts list is less than 2 then

PRINT “Error: Invalid format. Every line must have a course number and a title.”

EXIT program

CREATE a new object called courseDetails to store course information

SET courseDetails.courseNumber to parts[0]

SET courseDetails.courseTitle to parts[1]

CREATE a new list inside courseDetails called prerequisites

IF the number of parts in the parts list is more than 2

SET a counter i to 2

WHILE i is less than the total number of parts

ADD parts[i] to courseDetails.prerequisites

INCREASE i by 1

INSERT courseDetails into abcTree using the KEY courseDetails.courseNumber

FOR each course in abcTree

FOR each prereq in course.prerequisites

FIND lookupResult in abcTree with KEY prereq

IF lookupResult is not found THEN

PRINT “Error: Prerequisite” + “ for course ” + course.courseNumber + “ is not a valid course.”

EXIT program

END IF statement

PRINT “All courses have been loaded.”

IF the user entered “3”

PRINT “Please enter a course number: ”

GET searchTerm (input)

FIND course in abcTree WITH KEY searchTerm

IF a course is found then

PRINT course.courseNumber, course.courseTitle

PRINT “Prerequisites:”

FOR EACH prereq in course.prerequisites

PRINT prereq

ELSE

PRINT “Sorry, course was not found.”

END search

IF the user entered “2”

PRINT “Sorted course list: ”

FOR every course in abcTree

PRINT course.courseNumber, course.courseTitle

Vector Runtime Analysis

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| While there are still lines left to read in the file | 1 | n | n |
| Read the next file in the line | 1 | n | n |
| Split the line into parts using the comma as a divider | 1 | n | n |
| Create a new object called courseDetails | 1 | n | n |
| Set courseDetails.courseNumber to parts[0] | 1 | n | n |
| Set courseDetails.courseTitle to parts[1] | 1 | n | n |
| Create a new list inside courseDetails called prerequisites | 1 | n | n |
| If the number of parts in the parts list is less than 2 | 1 | n | n |
| ADD the courseDetails object to the allCourses list | 1 | n | n |
| **Total Cost** | | | 9n |
| **Runtime** | | | O(n) |

Hash Table Runtime Analysis

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| While there are still lines left to read in the file | 1 | n | n |
| Read the next file in the line | 1 | n | n |
| Split the line into parts using the comma as a divider | 1 | n | n |
| Create a new object called courseDetails | 1 | n | n |
| Set courseDetails.courseNumber to parts[0] | 1 | n | n |
| Set courseDetails.courseTitle to parts[1] | 1 | n | n |
| Create a new list inside courseDetails called prerequisites | 1 | n | n |
| Store courseDetails in courseHashTable with key courseDetails.courseNumber | 1 | n | n |
| Add courseDetails to the courseList | 1 | n | n |
| For each currentCourse in courseList | 1 | n | n |
| For each prereq in currentCourse.prerequisites | 1 | n \* p | n \* p |
| Find lookupResult in courseHashTable with key prereq | 1 | n \* p | n \* p |
| **Total Cost** | | | 9n + 2np |
| **Runtime** | | | O(n \* p) |

Binary Search Tree Runtime Analysis

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| While there are still lines left to read in the file | 1 | n | n |
| Read the next file in the line | 1 | n | n |
| Split the line into parts using the comma as a divider | 1 | n | n |
| Create a new object called courseDetails | 1 | n | n |
| Set courseDetails.courseNumber to parts[0] | 1 | n | n |
| Set courseDetails.courseTitle to parts[1] | 1 | n | n |
| Create a new list inside courseDetails called prerequisites | 1 | n | n |
| Insert courseDetails into abcTree using the key courseDetails.courseNumber | O(log n) | n | n log n |
| For each course in abcTree | 1 | n | n |
| For each prereq in course.prerequisites | 1 | n \* p | n \* p |
| Find lookupReselt in abcTree with key prereq | O(log n) | n \* p | n \* p \* log n |
| **Total Cost** | | | 7n + n log n + n p log n |
| **Runtime** | | | O(n p log n) |

Advantages and Disadvantages

The vector data structure is very simple and straightforward. Its structure is streamlined, which makes it very quick to sort for printing the whole list. Some cons would be that it doesn’t accomplish as complex of activities as the other two, and it takes longer to search for a specific course because it has to loop through and check each course in the catalog individually. We can see the evidence of this in the runtime analysis that its runtime is O(n), which is the slowest of the three. This isn’t a big problem with a manageable number of courses, but if there were hundreds or thousands of courses, it would create quite the issue. The hash table is the fastest of the three when it comes to searching for a course, but it is slower when printing the sorted list because it has to take extra steps to sort the random data. Looking at the table, the tree data is in the middle of the two for speed when searching for a course. It is also pretty fast for searching for a specific course, but it is better than the hash table at sorting because it is more organized. A con is that the inserting is a several step process, so the code is more complex than the other two.   
  
 Recommendation

My recommendation would be to utilize the binary search tree in the final program for the ABCU academic advisors. This is because this data structure is fast for searching and printing the entire sorted list.