Emily Wood  
CS 300  
Project One  
  
**Resubmit pseudocode from previous pseudocode assignments and update as necessary**.

LOAD Courses  
validFile = TRUE  
i = 0  
WHILE (Not end of file)  
 courseInformation[] = GETLINE(file, delimeter = “,”)  
 APPEND  
 IF (sizeof(courseInformation) < 2)  
 validFile = FALSE  
 BREAK  
 END IF  
   
 courseNumber[i] = courseInformation[0]  
 courseName[i] = courseInformation[1]   
 ++i  
  
 IF (sizeof(courseInformation) > 2)  
 j = 0  
 FOR (k=2; k < sizeof(courseInformation); ++ k)  
 prerequisiteCourses[j] = courseInformation[k]  
 ++j  
 END FOR  
 END IF  
END WHILE  
  
IF (validFile = TRUE)  
 FOR (course in prerequisiteCourses)  
 IF (course is not in courseNumbers)  
 validFile = FALSE  
 BREAK  
 END IF  
 END FOR  
END IF  
  
Return validFile

Vector<Course>  
  
Course courses  
  
WHILE (not end of file)  
 courseInformation[] = GETLINE(file, delimeter = “,”)  
 APPEND  
   
 courses.number = courseInformation[0]  
 courses.name = courseInformation[1]  
  
 IF (sizeof(courseInformation > 2)  
 FOR (i = 2; i < sizeof(courseInformation); ++i)  
 courses.prerequisite = courseInformation[i]  
 END FOR  
 END IF  
END WHILE

INPUT courseNumber

IF (courseNumber = courses.courseNumber)  
 printCourseInformation(Vector<Course> courses, courseNumber)

ELSE  
 OUTPUT “Not a Valid Course”  
END IF

IF (currentNode is not NULL)

Create new NODE

INSERT node at beginning

END IF

ELSE IF (currentNode points to key)

currentNode pointing to key = key

currentNode pointing to bid = bid

next Node = NULL

END IF

ELSE

WHILE (currentNode pointing to next = NULL)

currentNode = currentNode pointing to NEXT

END WHILE

next Node = new Node

END ELSE

FOR (begin nodes to end nodes)

IF (current node pointing to key does not equal max)

OUTPUT current node pointing to key

OUTPUT current node pointing to bidId

OUTPUT current node pointing to title

OUTPUT current node pointing to amount

OUTPUT current node pointing to fun

END IF

Node becomes next node

END FOR

WHILE (not end of file)  
 courseInformation[] = GETLINE(file, delimeter = “,”)  
 APPEND  
   
 courses.number = courseInformation[0]  
 courses.name = courseInformation[1]  
  
 IF (sizeof(courseInformation > 2)  
 FOR (i = 2; i < sizeof(courseInformation); ++i)  
 courses.prerequisite = courseInformation[i]  
 END FOR  
 END IF  
END WHILE

addNode(Node\* node, Course courses)  
 IF (node->courses.prerequisite > courses.prerequisite)  
 IF (node->left is NULL)  
 node->left = new Node(courses)  
 END IF  
 ELSE  
 addNode(node->left, courses)  
 END ELSE  
 END IF  
 ELSE  
 IF (node->right is not NULL)  
 node->right = new Node(courses)  
 END IF  
 ELSE   
 addNode(node->right, courses)  
 END ELSE  
 END ELSE  
END addNode

inOrder(Node\* node)  
 IF (node is not NULL)  
 inOrder(node->left)  
 OUTPUT courses.number  
 OUTPUT courses.name  
 OUTPUT courses.prerequisites  
 inOrder(node->right)  
 END IF  
END inOrder

**Create pseudocode for a menu**.

int choice = 0  
WHILE (choice does not equal 9)  
 OUTPUT “Welcome to the course Planner”  
 OUTPUT “1. Load Data Structure.”  
 OUTPUT “2. Print Course List.”  
 OUTPUT “3. Print Course.”  
 OUTPUT “9. Exit.”

INPUT choice  
  
 IF (choice does not equal 1,2,3,9)  
 OUTPUT choice  
 OUTPUT “ is not a valid option.”  
 continue  
 END IF  
  
 SWITCH (choice)  
 CASE 1  
 courses = loadCourses(file)  
 break  
 CASE 2  
 sortCourses(courses, 0, courses.size() – 1)  
 printCourseList(courses)  
 break  
 CASE 3  
 OUTPUT “What course would you like to know about?”  
 INPUT inputCourse  
 printCourseInfo(inputCourse)  
 break  
 END SWITCH  
OUTPUT “Thank you for using the course planner!"  
  
**Design pseudocode that will print out the list of the courses in the Computer Science program in alphanumeric order.  
  
partition(bids, begin, end)** int low = begin  
 int high = end  
 int midpoint = low + ((high – low) / 2)  
 string pivot = courses[midpoint].name  
 bool done = false  
 WHILE (done is false)  
 WHILE (courses[low].name < pivot)  
 low = low + 1  
 END WHILE  
 WHILE (pivot < courses[high].name)  
 high = high – 1  
 END WHILE  
 IF (low >= high)  
 done = true  
 END IF  
 ELSE  
 string temp = courses[low].name  
 courses[low].name = courses[high].name  
 courses[high].name = temp  
 low = low + 1  
 high = high – 1  
 END ELSE  
 END WHILE  
END partition **sortCourses(courses, begin, end)** int mid = 0  
 IF (begin >= end)  
 return  
 END IF  
 mid = partition(courses, begin, end)  
 sortCourses(bids, begin, mid)  
 sortCourses(bids, mid + 1, end)  
END sortCourses

printCourseList(courses)  
 FOR (int i = 0; i < courses.size(); ++i)  
 OUTPUT courseNumber[i]  
 OUTPUT courseName[i]  
 END FOR  
END printCourseList  
  
**Evaluate the run-time and memory of data structures that could be used to address the requirements**.

|  |  |  |  |
| --- | --- | --- | --- |
| **Vector** | | | |
| **CODE** | **LINE COST** | **# OF 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** | **# OF 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 | n | n |
| For each prerequisite of the course | 1 | n | n |
| Print the prerequisite course information | 1 | n | n |
| Total Cost | | | 5n |
| Runtime | | | O(n) |

|  |  |  |  |
| --- | --- | --- | --- |
| **Tree** | | | |
| **CODE** | **LINE COST** | **# OF 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 | n | n |
| For each prerequisite of the course | 1 | n | n |
| Print the prerequisite course information | 1 | n | n |
| Total Cost | | | 5n |
| Runtime | | | O(n) |

**Explain the advantages and disadvantages of each structure in your evaluation.**

**Binary trees can be used to reflect relationships between data, and they can store an arbitrary number of data values. They can be limited in that deleting nodes is a complex procedure and insertion, deletion, and search operations are dependent on the height of the tree.**

**With hash tables, insert, delete, and search operations are very fast and can be done in O(1) time Hash tables can also store large amounts of data. Hash functions, however, tend to produce duplicate keys, which cause collisions.**

Vectors are at an advantage because elements can be inserted and deleted easily, and multiple objects can be stored. Vectors are at a disadvantage because memory consumption is greater, and it is not indexed.

**Make a recommendation for which data structure you will plan to use in your code**.

Given that all the data structures have a notation of O(n), their efficiencies are the same. I would recommend using a vector in this scenario. Given that the data is small, the search time will be negligible to warrant using a tree, which is faster in the search. Its ease of use for storage is my key decision maker.