Griffin Hood

CS-300

Project 1

April 2, 2023

***Design pseudocode to define how the program opens the file, reads the data from the file, parses each line, and checks for file format errors***

***&***

***Design pseudocode to show how to create course objects and store them in the appropriate data structure***

(I sort of combined these two parts to eliminate another loop through the prerequisites to create objects)

LOAD the CSV file

GET all of the lines from the CSV file and store in an array called lines

DECLARE an array prerequisitesToValidate

DECLARE a HASH TABLE called courses where the keys are course ids and the values are objects containing the course information

FOREACH line in LINES

SPLIT the line around each comma into an array

IF the length of the array is less than 2

THROW and error and explain the validation rules

SET courseId equal to the first element in the array courseInfo

SET courseName equal to the second element in the array

DECLARE and array called coursePrerequisites

FOR EACH item in courseInfo, starting at the third element

ADD the value (the prerequisite) to coursePrequisites

IF the value of the current element DOES NOT exists as a key in the HASH TABLE

ADD the value to prerequisitesToValidate

CREATE an object of type Course with the arguments courseId, courseName, and coursePrerequisites

STORE the object in the HASH TABLE with they key being the courseId

// Validate each prereq that couldn’t be validated before

FOREACH item in prerequisitesToValidate

IF item is not a key in courses

THROW an error (there is a prerequisite that isn’t also on its own line)

**VECTOR**

CREATE a vector called finalVector

ADD each value from the HASH TABLE to finalVector

**BINARY TREE**

CREATE a binary tree called tree

Insert each value from the HASH TABLE into tree following the binary search algoritm

***Design pseudocode that will search the data structure for a specific course and print out course information and prerequisites***

void printCourseInformation(Vector<Course> courses, String courseNumber)

FOREACH Course object in the courses

IF the course number of the course object is equal to courseNumber

DISPLAY the course number

DISPLAY the course name

FOREACH prerequisite in the Course object

DISPLAY the prerequisite

**2. MENU PSEUDOCODE**

SET the variable choice equal to null

WHILE choice is not equal to 4

DISPLAY the menu

READ input from the user’s keyboard and store in choice

IF choice is equal to 1

LOAD the data using the algorithm above

ELSE IF choice is equal to 2

DISPLAY all courses using the algorithm above

ELSE IF choice is equal to 3

SEARCH for the specific course ID in the data structure

DISPLAY that course only

ELSE IF choice is equal to 4

BREAK

ELSE

CONTINUE

**3. PSEUDOCODE to display a list of the courses in the Computer Science program in alphanumeric order**

**mergeSort(vector courses, int low, int high)**

IF low is greater than high

RETURN

SET mid equal to low + high divided by 2

CALL mergeSort(courses, low, mid)

CALL mergeSort(courses, mid+1, high)

MERGE the vector of courses back together in order

**printCourses(vector courses)**

FOR EACH course in courses

DISPLAY course information to the console

FOR EACH prerequisite of course

DISPLAY each prerequisite to the console

**EVALUATION**

|  |  |  |  |
| --- | --- | --- | --- |
| **CODE** | **LINE COST** | **# TIMES EXECUTED** | **TOTAL COST** |
| FOREACH line in LINES | **1** | **n** | **n** |
| SPLIT the line around each comma into an array | **1** | **n** | **n** |
| IF the length of the array is less than 2 | **1** | **n** | **n** |
| THROW and error and explain the validation rules | **1** | **1** | **1** |
| SET courseId equal to the first element in the array courseInfo | **1** | **n** | **n** |
| FOR EACH item in courseInfo, starting at the third element | **1** | **n** | **n** |
| ADD the value (the prerequisite) to coursePrequisites | **1** | **n** | **n** |
|  |  |  |  |
|  |  | **TOTAL COST** | **6n+1** |
|  |  | **RUNTIME** | **O(6n+1) = O(n)** |
|  |  |  |  |

**Vector**

* Advantages
  + Simple to iterate over
  + Easy to delete or add an element
* Disadvantages
  + Hard to visualize relationships between data
  + Memory is allocated dynamically which can be slow

**Hash Table**

* Advantages
  + Insert and remove operations are very fast (O(1))
  + Can easily store very large amounts of data
* Disadvantages
  + Collisions can happen
  + Good hash functions that generate unique keys can be expensive

**Binary Tree**

* Advantages
  + Very easy to visualize relationships among the data
  + Limit less in the number of nodes that can be stored in a tree
* Disadvantages
  + Removing a node from the tree can be an expensive operation
  + Most operations are dependent on the height of the tree

Based on the above analysis, I would recommend using the hash table data structure. The has table is a simple structure that is very fast when accessing, adding, or deleting data from the structure. The hash table has lookup times of an average of O(1) or constant time. While vectors and trees take O(n) or linear time. In other words, those two structures depend on how many elements are in the structure. Additionally, if the number of entries of a hash table is known in advance, this makes it even faster to access, insert, and remove data.