Christopher Anderson

Prof. Ostrowski

Data Structures and Algorithms: Analysis and Design

6-2 Assignment

4/4/2023

Project One

Pseudocode

// Vector functions

START Function named Partition RETURNING integer with PARAMETERS of vector of Course objects named courseVector, integer named begin, integer named end

INITIALIZE integer named low to begin and integer high to end

INITIALIZE integer named middle to middle index of courseVector

INITIALIZE string named pivot to courseNumber of object at middle index of courseVector

INITIALIZE bool named done to FALSE

WHILE done is FALSE

WHILE courseNumber at low index in bids is LESS THAN pivot

INCREMENT low

WHILE pivot is LESS THAN couseNumber at high index of courseVector

DECREMENT high

IF low is GREATER THAN OR EQUAL to high

SET done to TRUE

ELSE

INITIALIZE Course named temp to low index of courseVector

SET low index of courseVector to high index of courseVector

SET high index of courseVector to temp

INCREMENT low

DECREMENT high

RETURN high

END

START Function named quicksort of void RETURN with PARAMETERS of Vector of Course objects named courseVector, integer named begin, and integer named end

INITIALIZE integer named middle to 0

IF begin’s Course object’s name GREATER THAN OR EQUAL TO end’s Course object’s name

RETURN

SET middle to return value of partition function with courseVector, end, and begin as parameters

CALL quicksort with courseVector, begin, and middle as parameters

CALL quicksort with courseVector, middle + 1, and end as parameters

END

START Function named selectionSort of void RETURN with PARAMETER of vector of Course objects named courseVector

DEFINE integer named min

INITIALIZE integer named size\_t to number of objects in courseVector

INITIALIZE integer named pos to 0

SET min to pos

FOR pos EQUAL TO 0, but LESS THAN size\_t – 1, INCREMENT pos each loop cycle

IF courseNumber of object in courseVector at pos index is LESS THAN courseNumber of object in courseVector at min index

SWAP Course object at pos index with Course object at min index

SET min to pos

END

START loadCourseVector function of void return with Vector of Course objects reference named courseVector and string named DataFilePath as parameter

INITIALIZE file in-stream named DataInHandle to file associated with DataFilePath

TEST the following statements for error codes

FOR each row in the file

DEFINE a Course named newCourse

INITIALIZE a string named fileRow to the current row

SECTION fileRow into three strings by the first 2 occurrences of the ‘,’ char

SET newCourse’s courseNumber member to the first section of fileRow excluding the first occurrence of ‘,’

SET newCourse’s name member to the second section of fileRow, between the first and second occurrences of ‘,’

ADD 1st prerequisite as head of prerequisiteList for newCourse object

IF fileRow INCLUDES additional occurrences of ‘,’

FOR each additional occurrence of ‘,’ + 1

SET (n-1)th prerequisite’s next pointer to the nth prerequisite

PUSH newCourse to courseVector

ERROR is caught

OUTPUT error code to console

STOP

START function named printVector of void return with parameter of Vector of Course objects reference named courseVector

INITIALIZE iterator for course objects named courseIterator to courseVector’s begin flag

WHILE courseIterator IS NOT EQUIVALENT to courseVector’s end flag

OUPUT courseIterator’s current objects courseNumber, “, “, courseIterator’s name member

CALL printCoursePrerequisiteList with courseIterator’s current object as parameter

OUTPUT newline

INCREMENT courseIterator

STOP

START printSortedVector function of void return with vector of Course objects named courseVector

INITIALIZE integer vectorSize to return of courseVector’s size function

CALL quicksort with courseVector, 0 and vectorSize as parameters

CALL printVector with courseVector as parameter

STOP

START printObjectVector function of void return with string named courseNumberKey and vector of Course objects named courseVector as parameters

INITIALIZE iterator of Course objects named courseIterator to courseVector’s begin flag

WHILE courseIterator IS NOT EQUIVALENT to courseVector’s end flag

IF courseIterator’s current object’s courseNumber IS EQUIVALENT to courseNumberKey

OUTPUT courseIterator’s courseNumber, “, “, courseIterator’s name, “, “

CALL printCoursePrerequisiteList with courseIterator’s current object as parameter

STOP

// Hash Table functions

START loadCourseInfoHash function of void return with HashTable reference named courseHashTable and string named hashDataFilePath as parameter

INITIALIZE file in-stream named hashDataInHandle to file associated with hashDataFilePath

TEST the following statements for error codes

FOR each row in the file

DEFINE a Course named newCourse

INITIALIZE a string named fileRow to the current row

SECTION fileRow into three strings by the first 2 occurrences of the ‘,’ char

SET newCourse’s courseNumber member to the first section of fileRow excluding the first occurrence of ‘,’

SET newCourse’s name member to the second section of fileRow, between the first and second occurrences of ‘,’

ADD 1st prerequisite as head of prerequisiteList for newCourse object

IF fileRow INCLUDES additional occurrences of ‘,’

FOR each additional occurrence of ‘,’ + 1

SET (n-1)th prerequisite’s next pointer to the nth prerequisite

INITIALIZE Node named newNode to newCourse as the Course member and the return of the hash function with the integer converted from newCourse’s courseNumber element as the key

ADD newNode to courseHashTable’s Node vector

ERROR is caught

OUTPUT error code to console

STOP

START printAllHashTable function of void return with HashTable named courseHashTable as parameter

INITIALIZE Node pointer named nextNode to null

FOR iterator named nodeIterator INITIALIZED to the begin flag of courseHashTable’s node vector, nodeIterator IS NOT EQUIVALENT to node vector’s end flag, INCREMENT nodeIterator each loop cycle

IF nodeIterator’s key IS NOT UINT\_MAX

OUTPUT “Course Number: “, nodeIterator’s Course’s courseId, “\nCourse Name: “, nodeIterator’s Course’s courseName, newline

OUTPUT “Prerequisites: “

CALL printCoursePrerequisiteList function’s return with parameter of nodeIterator’s current Node’s Course object

OUTPUT newline

STOP

START printCoursePrerequisiteList function of void return with parameter of Course object named aCourse

INITIALIZE integer named counter to 1

INITIALIZE Node pointer named prerequisiteNode to prerequisiteList’s head

OUTPUT prerequisiteNode’s prerequisiteNumber member

SET prerequisiteNode to prerequisiteNode’s next object

WHILE counter is LESS THAN aCourse’s prerequisiteList’s size member

OUTPUT “, “, prerequisiteNode’s prerequisiteNumber member

SET prerequisiteNode to prerequisiteNode’s next object

INCREMENT counter

STOP

START printObjectHashTable of void return with parameter of string named courseNumberKey and HashTable object named hashTable

INITIALIZE integer named hashKey to return of hashTable’s hash function with courseNumberKey as parameter

FOR iterator named nodeIterator INITIALIZED to the begin flag of hashTable’s node vector, nodeIterator IS NOT EQUIVALENT to node vector’s end flag, INCREMENT nodeIterator each loop cycle

IF nodeIterator’s key IS EQUIVALENT to hashKey

OUTPUT nodeIterator’s current Node’s Course object’s courseNumber, “, ”, nodeIterator’s name, “, “

CALL printCoursePrerequisiteList with nodeIterator’s course as parameter

STOP

START printSortedHashTable function of void return with HashTable named courseHashTable

INITIALIZE integer hashTableSize to courseHashTable’s tableSize

CALL quicksort with courseHashTable’s Node vector, 0 and hashTableSize as parameters

CALL printVector with courseHashTable’s Node vector as parameter

STOP

// Binary Search Tree functions

START Insert function of void return with course named newCourse as parameter

INITIALIZE Node pointer named currentNode to null

IF root IS EQUIVALENT to null

SET root’s course to newCourse

SET root’s left pointer to null

SET root’s right pointer to null

ELSE

SET currentNode to root

WHILE currentNode IS NOT null

IF newCourse’s courseNumber IS LESS THAN currentNode’s courseNumber

IF currentNode’s left pointer IS EQUIVALENT to null

SET currentNode’s left pointer’s course to newCourse

SET currentNode to null

ELSE

SET currentNode to currentNode’s right pointer

ELSE

IF currentNode’s right pointer IS EQUIVALENT to null

SET currentNode’s right pointer’s course to newCourse

SET currentNode to null

ELSE

SET currentNode to currentNode’s right pointer

STOP

START loadCourseInfoBST function of void return with BinarySearchTree reference named courseBST and string named dataFilePath as parameter

INITIALIZE file in-stream named dataInHandle to file associated with dataFilePath

TEST the following statements for error codes

FOR each row in the file

DEFINE a Course named newCourse

INITIALIZE a string named fileRow to the current row

SECTION fileRow into three strings by the first 2 occurrences of the ‘,’ char

SET newCourse’s courseNumber member to the first section of fileRow excluding the first occurrence of ‘,’

SET newCourse’s name member to the second section of fileRow, between the first and second occurrences of ‘,’

ADD 1st prerequisite as head of prerequisiteList for newCourse object

IF fileRow INCLUDES additional occurrences of ‘,’

FOR each additional occurrence of ‘,’ + 1

SET (n-1)th prerequisite’s next pointer to the nth prerequisite

CALL Insert function with newCourse as parameter

ERROR is caught

OUTPUT error code to console

STOP

START printCoursePrerequisiteList function of void return with parameter of Course object named aCourse

INITIALIZE integer named counter to 1

INITIALIZE Node pointer named prerequisiteNode to prerequisiteList’s head

OUTPUT prerequisiteNode’s prerequisiteNumber member

SET prerequisiteNode to prerequisiteNode’s next object

WHILE counter is LESS THAN aCourse’s prerequisiteList’s size member

OUTPUT “, “, prerequisiteNode’s prerequisiteNumber member

SET prerequisiteNode to prerequisiteNode’s next object

INCREMENT counter

STOP

START recursiveNodePrint of void return with parameter of Node pointer named aNode

IF aNode IS NOT null

CALL recursiveNodePrint with aNode’s left pointer as parameter

OUTPUT courseNumber, “, “, name, “, “

CALL printCoursePrerequisiteList with aNode’s course as parameter

CALL recursiveNodePrint with aNode’s right pointer as parameter

STOP

START printAllCourseBST of void return with parameter of BinarySearchTree name bst

INITIALIZE Node pointer named currentNode to bst’s root

CALL recursiveNodePrint with currentNode as parameter

STOP

START printObjectBST of void return with parameter of string named courseNumberKey and BinarySearchTree object reference named bst

INITIALIZE new Course object named aCourse to the return value of bst’s Search function with courseNumberKey as parameter

IF aCourse courseNumber IS EQUIVALENT to courseNumberKey

OUTPUT aCourse’s courseNumber, “, “, aCourse’s name, “, “

CALL printCoursePrerequisiteList with aCourse as parameter

ELSE

OUTPUT error code for courseNumber not found

STOP

START menu function returning bool with *desired data structure* named dds, string named filePath, and bool named exitCondition as parameter

OUTPUT “MENU-“, newline, “1.Load Data Structure:\n2.Print Course List:\n3.Print Course:\n4.Exit\n”

INITIALIZE short integer named userIn

SWITCH controlled by userIn

in CASE userIn is 1

CALL loadCourseVector, loadCourseInfoHash, or loadCourseInfoBST function with dds and filePath as parameters

In CASE userIn is 2

CALL printVector, printAllHashTable, or printAllCourseBST function with dds as parameter

In CASE userIn is 3

OUTPUT “Enter course number: “

INITIALIZE string named courseNumberInput to USER’S INPUT

CALL printObjectVector, printObjectHashTable, or printObjectBST with dds and courseNumberInput

In CASE userIn is 4

SET exitCondition to TRUE

DEFAULT case

OUTPUT “Invalid input—“, newline, userIn, newline

STOP

Evaluation

In this evaluation of the pseudocode for the academic advisors in the Computer Science department at ABCU I will be comparing the data structures at hand, and I will explain my recommendation for the Binary Search Tree as the data structure to be utilized for this software package. When selecting the best data structure for this software package it is vital to consider differences in the run-time time and memory consumption of the available data structures. Based on the advisor’s requirements, an understanding of the values and limitations of each data structure is necessary to allow this project to reach it’s full potential. After conducting a full analysis of the system requirements and the data structures at hand I will recommend the Binary Search Tree be implemented as the software package’s primary data structure. Throughout the analysis of this pseudocode, I will explore the strengths and weaknesses of implementing the Vector, Hash Table and Binary Search Table data structures.

During the selection for the data structure to be utilized for the academic advisors’ software package, the greatest considerations are run-time effectiveness and memory consumption of each structure. When comparing the execution time to load a file into a Vector, Hash Table or Binary Search Tree it can become difficult to decern the best data structure, since all three data structures evaluate to O(N), O(N log N) and O(N) for the vector load, binary search tree load and hash table load functions, respectively. But I have determined that the Binary Search Tree is most effective for creating a sorted database, that includes natural security since it is a list of object pointers, and it will minimize run-time for retrieval of members with a standard non-linear search pattern. The paramount concerns when determining the data structures, I will include in my software packages are run-time efficiency and memory requirements.

There are several advantages and disadvantages to the implementation of Vectors, Hash Tables and Binary Search Tree data structures. For the Vector data structure, it is known to be stable and consistent design but searching for specific members often requires extra steps to sort the structure allowing for faster run-time execution and this structure includes only basic security features. Given a need for heightened security, with the addition of the hash key and hash function, the Hash Table becomes a viable option since this structure contains in it’s class declaration a vector of Node objects it does have all the capacities of a standard vector but with the addition of fast access through the usage of hash keys that also act as a security measure. The data structure that I am recommending for this project, the Binary Search Tree has the advantages of sorting itself as it loads a file to the structure, sorting a new object as it’s added, along with decreased run-time when searching for a specific object since the worst case scenario will only require the search function to traverse each level of the binary tree structure, and finally the data stored in this structure has higher levels of security since the structure utilizes pointer Nodes for its list that creates difficulty for intruders to predict the memory addresses of the data being stored. The Binary Search Tree, Vector and Hash Table data structures each have ingrained inadequacies and dominions that necessitate consideration when developing a software package.

I evaluated the pseudocode for the academic advisors in the Computer Science department at ABCU and with careful consideration I determined that the Binary Search Tree is the best data structure for this software package. The run-times related to these data structures during operations and the memory requirements for Vectors, Hash Tables and Binary Search Trees are pivotal factors in my decision. Based upon the academic advisor’s system requirements, I can isolate the values and limitations that each data structure will bring to this project. During this pseudocode analysis I investigated the implications of developing the Computer Science department’s software package with a Vector, Binary Search Table or Hash Table as the primary data structure utilized.