Jacob Byars

4/9/2023

CS-300-J7682 DSA: Analysis and Design 23EW

Project 1

Pseudocode

**PART 1 (include pseudo code with changes from previous milestones)**

**VECTOR**

**Prompt 1**

FUNCTION getCourses(string fileNameWithPath)

DECALRE courseVector as a new Vector<Course>()

SET userFile equal to open(fileNameWithPath)

IF userFile equals -1 then  
 OUTPUT “404 – File Not Found”

RETURN courseVector

END IF

SET nextLine equal to “Start”

WHILE nextLine is not equal to EOF THEN

SET nextLine equals userFile.readLine()

SET tokenList equal to the split of nextLine on “,”

SET tokenSize equal to the size of tokenList

IF tokenSize < 2 Then

OUTPUT “Error: Not enough parameters on line.”

CONTINUE to next line

ELSE IF tokenList[2] is not equal to “” AND courseTree.contains(tokenList[2]) is equal to false THEN

OUTPUT “Error: prerequisite course not found in tree.”

CONTINUE to next line

ELSE

SET newCourse equal to a newCourse(tokenList[0], tokenList[1], tokenList[2])

ADD newCourse to courseTree

END IF

END WHILE

END FUNCTION

**Prompt 2**

CLASS Course

DECLARE courseNumber as string

DECLARE courseTitle as string

DECLARE prerequisites[] as vector list of strings

CONSTRUCTOR Course(readLine)

SET courseNumber equal to SPLIT(readLine, DELIMETER equal to , )[0]

SET courseTitle equal to SPLIT(readLine, DELIMETER equal to , )[1]

IF length of SPLIT(readLine, DELIMETER equal to , ) is greater than 2 THEN

SET prerequisites equal to SPLIT(readLine)[2 to length of SPLIT(readLine, DELIMETER equal to , )]

END IF

END CONSTRUCTOR

END CLASS

FUNCTION createCourseObject(Courses <Course>, File userFile)

SET userLines[] equal to “ “

IF readFile(userFile, userLines) equals TRUE THEN

FOR each line in userLines

APPEND new Course(line) To COURSES

END FOR

ELSE

OUTPUT “File cannot be read”

END IF

END FUNCTION

**Prompt 3**

FUNCTION printCourseInformation (Vector<Course> courseList, String courseNumber)

FOR each course in courseList

IF course[0] is equal to courseNumber THEN

OUTPUT “Course Number is: “ AND course[0] AND new line

OUTPUT “Course Name is: “ AND course[1] AND new line

OUTPUT “Prerequisites are: “

SET courseSize equal to course.size()

IF courseSIze < 2 THEN

OUTPUT “NONE”

ELSE

FOR (x = 2; x<courseSize; ++x)

IF x equals courseSize -1 THEN

OUTPUT course[x] AND new line

ELSE

OUTPUT course[x] AND “, “

END IF

END FOR

END IF

OUTPUT new line

END IF

END FOR

END FUNCTION

**Hash Table**

**Prompt 1**

FUNCTION readFile(File userFile, lines[])

DECLARE courseNumber[] list

DECLARE courseTitles[] list

DECLARE prerequisites[] list

DECLARE line

SET i equal to 0

SET j equal to 0

SET flag equal to TRUE

WHILE not EOF of userFile

SET courseINFO[] equal to SPLIT (READLINE(userFile, line), DELIMETER = ,)

APPEND line to lines

IF length of courseInfo is less than 2 THEN

SET flag equal to FALSE

BREAK

END IF

SET courseNumber[i] equal to courseInfo[0]

SET courseTitles[i] equal to courseInfo[1]

ADD 1 to i

IF length of courseInfo is greater than 2 THEN

FOR k equals 2 to LENGTH of courseInfo

Set prerequisites[j] = courseInfo[k]

ADD 1 to j

END FOR

END IF

END WHILE

IF flag is equal to TRUE THEN

FOR each p in prerequisites

IF p not in courseNumbers

SET flag equal to FALSE

BREAK

END IF

END FOR

END IF

RETURN flag

END FUNCTION

**Prompt 2**

CLASS Course

DECLARE courseNumber as string

DECLARE courseTitle as string

DECLARE prerequisites[] as list of strings

CONSTRUCTOR Course(readLine)

SET courseNumber equal to SPLIT(readLine, DELIMETER equal to , )[0]

SET courseTitle equal to SPLIT(readLine, DELIMETER equal to , )[1]

IF length of SPLIT(readLine, DELIMETER equal to , ) is greater than 2 THEN

SET prerequisites equal to SPLIT(readLine)[2 to length of SPLIT(readLine, DELIMETER equal to , )]

END IF

END CONSTRUCTOR

END CLASS

FUNCTION createCourseObject(Courses <Course>, File userFile)

SET userLines[] equal to “ “

IF readFile(userFile, userLines) equals TRUE THEN

FOR each line in userLines

APPEND new Course(line) To COURSES

END FOR

ELSE

OUTPUT “File cannot be read”

END IF

END FUNCTION

**Prompt 3**

FUNCTION MAIN()

INPUT filename

SET userFile equal to new FILE(filename)

Declare Courses <Course> as vector

Call createCourseObject(Courses, userFile)

INPUT courseNumber

IF Course is empty THEN

OUTPUT “No objects read from the file”

ELSE

OUTPUT course information with pre requisites.

END IF

END FUNCTION

**Binary Tree**

**Prompt 1**

FUNCTION readFile(File userFile)

IF OPEN userFile equals -1 THEN

RETURN empty courseTree

END IF

DECLARE readLine

DECLARE new Course course

DECLARE empty CourseTree courseTree

SET readLine equal to userFile.readline()

WHILE readLine is not equal to EOF

Set courseInfo[] equal to SPLIT(readLine, Delimeter = “,”)

IF Length of courseInfo is less than 2 THEN

OUTPUT “Error not enough info for class”

CONTINUE

END IF

SET course->courseNum equal to courseInfo[0]

SET course->courseTitle equal to courseInfo[1]

IF length of courseInfo is greater than 2 THEN

FOR i = 2; i < courseInfo.size(); ++I;

CALL course.addPrerequestite(courseInfo[i], I -2, course, courseTree)

END FOR

END IF

ADD course to courseTree using the course->courseNum as the key

SET readLine equal to userFile.readline()

END WHILE

END FUNCTION

FUNCTION addPrereqesite(prerequisteId, key, userCourse, courseTree)

FOR each course in courseTree

IF course->courseId is equal to prerequisteID THEN

SET userCourse->prerequisite[userCourse->prerequisite.size-1] equal to prerequisteId

RETURN

END IF

END FOR

OUTPUT error prerequisite not found

RETURN

END FUNCTION

**Prompt 2**

CLASS Course

DECLARE courseNumber as string

DECLARE courseTitle as string

DECLARE prerequisites[] as vector list of strings

CONSTRUCTOR Course(readLine)

SET courseNumber equal to SPLIT(readLine, DELIMETER equal to , )[0]

SET courseTitle equal to SPLIT(readLine, DELIMETER equal to , )[1]

IF length of SPLIT(readLine, DELIMETER equal to , ) is greater than 2 THEN

SET prerequisites equal to SPLIT(readLine)[2 to length of SPLIT(readLine, DELIMETER equal to , )]

END IF

END CONSTRUCTOR

END CLASS

FUNCTION createCourseObject(Courses <Course>, File userFile)

SET userLines[] equal to “ “

IF readFile(userFile, userLines) equals TRUE THEN

FOR each line in userLines

APPEND new Course(line) To COURSES

END FOR

ELSE

OUTPUT “File cannot be read”

END IF

END FUNCTION

**Prompt 3**

FUNCTION printCourseInformation (CourseTree courseTree, String courseNumber)

FOR all userCourses in courseTree

IF userCourses->courseNumber equals courseNumber THEN

OUTPUT userCourse information

FOR each userCourses->prerequisite

OUTPUT prerequisite course information

END FOR

END IF

END FOR

END FUNCTION

**PART 2 (Main Menu Pseudocode)**

FUNCTION mainMenu()

SET userChoice equal to 0

WHILE userChoice is not 1, 2, 3, or 4 THEN

OUTPUT “Main MENU” and newline

OUTPUT “1) Load Data into DataStructure.” and newline

OUTPUT “2) Print Course List.” And new line

OUTPUT “3) Print Course.” And new line

OUTPUT “4) EXIT” and new line

OUTPUT “Please enter the number for the option you wish to do: ”

INPUT userChoice

END WHILE

SELECT userChoice

CASE 1

CALL loadDataStructure()

BREAK

CASE 2

CALL printCourseList()

BREAK

CASE 3

CALL printCourse()

Break

CASE 4

RETURN -1

CASE ELSE

OUTPUT “INVALID SELECTION. TRY AGAIN!” and new line

RETURN 0

BREAK

END SELECT

RETURN 0

END FUNCTION

**PART 3 – (print sorted Pseudocode)**

FUNCTION printSortedVector (Vector<course> courseList)

CALL sort(courseList.begin(), courseList.end())

FOR courses in courseList

OUTPUT “----------------------------------------------------------” and newline

OUTPUT “Course ID: “ and courses->courseID and newline

OUTPUT “Course Title: “ and course ->CourseTitle and newline

OUTPUT “Prerequisites: “ and newline

FOR prereq in course->prerequisite

OUTPUT prereq and new line

END FOR

END FOR

END FUNCTION

FUNCTION printSortedHashTable(HashTable\* courseList)

DECLARE course variable

SET key equal to hash(atoi(course.courseId.c\_str()));

FOR vector<Node>::iterator i = nodes.begin(); i != nodes.end(); ++I

IF i key does not equal UNIT\_MAX

OUTPUT course.courseID, course.courseTitle

FOR each preq in course->prerequisites

OUTPUT preq and newline

END FOR

SET node equal to i next

WHILE node is not equal to nullptr

OUTPUT node.courseID, node.courseTitle

FOR each preq in course->prerequisites

OUTPUT preq and newline

END FOR

END WHILE

END IF

END FOR

END FUNCTION

FUNCTION printSortedBinaryTree(CourseTree courseList)

CALL inOrder(courseList->root)

END FUNCTION

FUNCTION inOrder(TreeNode node)

IF node equals nullptr THEN

RETURN

END IF

CALL inOrder(node->left)

OUTPUT “--------------------------------” and newline

OUTPUT “Course ID: “ and node->courseID and newline

OUTPUT “Course Titel: “ and node->courseTitle and newline

OUTPUT “Prerequisites: “

FOR each prereq in node->prerequisites

OUTPUT prereq and newline

END FOR

CALL inOrder(node->right)

END FUNCTION

**PART 4 (Evaluation big O charts)**

**VECTOR**

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| Open File | 1 | 1 | 1 |
| While EOF not reached Read next line | 1 | N | N |
| If less than 2 values returned from split of line, then return error | 1 | N | N |
| Else continue | 1 | N | N |
| Initialize course vector | 1 | 1 | 1 |
| Loop over each line | 1 | N | N |
| Add course to vector | 1 | N | N |
| Pushback item | 1 | N | N |
| **Total Cost** | | | 6n+2 |
| **Runtime** | | | O(n) |

**Hash Table**

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| Open File | 1 | 1 | 1 |
| WHILE EOF not reached read next line | 1 | N | N |
| If less than 2 values returned from split of line, then return error | 1 | N | N |
| Else continue | 1 | N | N |
| Create HashTable | 1 | 1 | 1 |
| Insert course into hash table | 1 | N | N |
| Loop though File | 1 | N | N |
| For each line in file | 1 | N | N |
| Create temp course to hold values | 1 | N | N |
| **Total Cost** | | | 7n + 2 |
| **Runtime** | | | O(n) |

**Binary Tree**

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| Open File | 1 | 1 | 1 |
| While EOF not reached read next line | 1 | N | N |
| If less than 2 values returned from split of line, then return error | 1 | N | N |
| Else continue | 1 | N | N |
| Create Binary Search Tree | 1 | 1 | 1 |
| Insert course into tree | 1 | N | N |
| While Not EOF | 1 | N | N |
| Loop Though file | 1 | N | N |
| For each line in file | 1 | N | N |
| For first and second Values, ADD course id and Course Name | 1 | N | N |
| If any other values exist add prerequisite until newline found | 1 | N | N |
| **Total Cost** | | | 9n + 2 |
| **Runtime** | | | O(n) |

**Part 5 (Evaluation Analysis of different data structures)**

Vectors are arrays that can dynamically increase their size. This allows them to read and store the data quickly for the course information. The disadvantage of this is that searching the vector for a specific course is slower as it must search through the entire vector until the value is found.

Hash Tables are capable of quick searches as it stores the information by using a key. These keys map to information stored in the tables. In this case the key would be the course id. However the data is stored in an unsorted manner, and it can take a long time to go though the hash table and sort the courses so they were in alpha numeric order.

Binary Search trees main advantage is that its data is already stored in a semi sorted format. While it lacks the keys that make the search function of the hash tables so quick it is typically faster at searching than a Vector. The main disadvantage to a binary search tree is that any modifications to the tree may take a lot longer than the other data structures as it may need to modify each element in the tree.

**Part 6 (Evaluation Recommendation)**

My recommendation would be to use Vectors. Overall, I think Vectors would make the most sense as they have the shortest run time to insert new data. They are slower at searching since they have to loop over each element, and they do require some additional time to sort in alphanumeric sequence than a binary tree, but they would make the most sense as one of the main things is inputting the courses into the structure, and I think that compensates for the increased time taken to do the searches and sorting.