# Project 1

## Pseudocode

### Opening the File

//Opening the file function pseudocode

OPEN file.txt

ASSIGN coursesTree = new Tree<Courses>()

(FOR LOOP) each line in the file {

ASSIGN listEntries = split(line with comma)

(IF LOOP) listEntries is less than two {

RAISE ERROR message: there need to be at least two parameters on each line (course number and name)

CONTINUE

}

(IF LOOP) the course has a prerequisite validate if that course exists {

RAISE ERROR message if does not exist

CONTINUE

}

CREATE a new course with an adequate listEntries

APPEND the created course to the tree

}

### Vector pseudocode

OPEN the file function

int numPrerequisiteCourses(Vector<Course> courses, Course c) {

ASSIGN the variable totalPrerequisites to be equal to all the pre-requisites of “Course c”

(FOR LOOP) every pre-requisite located in the variable “totalPrerequisites” {

APPEND the values of pre-requisites into the variable “totalPrerequisites”

}

DISPLAY the number that represents all of the saved values in the variable “totalPrerequisites”

}

void printSampleSchedule(Vector<Course> courses) {

(FOR LOOP) every “Course c” and pre-requisites values saved in the vectors {

DISPLAY id and name of those”Courses c”

(IF LOOP) the saved “Course c” value in the vector has pre-requisites present {

(FOR LOOP) all the “Course c” pre-requisites bonded to those courses {

DISPLAY the pre-requisites next to the course

}

}

}

}

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

(FOR LOOP) all the “Course c” courses stored as vectors {

(IF LOOP) saved course name is the same as the course number {

DISPLAY the stored information of the courses

(FOR LOOP) go through the vector and for each pre-requisite binded to the course’s information {

DISPLAY all the pre-requisites saved within the course’s information

}

}

}

}

### Hashtable pseudocode

OPEN the file function

int numPrerequisiteCourses(Hashtable<Course> courses) {

ASSIGN the variable “totalPrerequisites” to be equal to the Hash-table of the prerequisites of the courses

(FOR LOOP) all the pre-requisites stored in the “totalPrerequisites” variable {

APPEND the values into the new “totalPrerequisites” Hash-table holding the pre-requisites

APPEND each course with key as ID and value as name of the course

}

DISPLAY the number that represents all of the saved values in the variable “totalPrerequisites”

}

void printSampleSchedule(Hashtable<Course> courses) {

(FOR LOOP) each of the key, value pairs assigned to all the courses {

DISPLAY the key of the course (Course ID) as the first value

(IF LOOP) there are pre-requisite values present within the course Hash-table {

(FOR LOOP) every pre-requisite value present within the course variable {

DISPLAY the pre-requisite information

}

}

}

}

void printCourseInformation(Hashtable<Course> courses, String courseNumber) {

(FOR LOOP) all the stored courses within the courses Hash-table {

(IF LOOP) the assigned course name equals the assigned couse number (course ID {

DISPLAY all the course’s info that is stored to the assigned name and number of that course;

(FOR LOOP) within the courses Hash-table search for every pre-requisite saved with the course {

DISPLAY all of the pre-requisites tied to the course they are stored in

}

}

}

}

### Tree pseudocode

OPEN the file function

int numPrerequisiteCourses(Tree<Course> courses) {

ASSIGN the variable “totalPrerequisites” to be equal to the left child of the Node

ASSIGN the variable “totalPrerequisites” to be equal to the right child of the Node

(FOR LOOP) all the pre-requisites stored within the variable “totalPrerequisites” {

APPEND left Node of the pre-requisites to “totalPrerequisites”

APPEND the right Node of the pre-requisites node to “totalPrerequisites”

}

DISPLAY the number that represents all of the saved values in the variable “totalPrerequisites”

}

void printSampleSchedule(Tree<Course> courses) {

(FOR LOOP) all of the nodes that equal to courses {

DISPLAY the name of the course after finding it

(IF LOOP) the binary search tree “Course” has a left node attached to it {

DISPLAY the attached left node and set it to be a prerequisite to the course

}

(IF LOOP) the binary search tree “Course” has a right node attached to it {

DISPLAY the attached right node and set it to be a prerequisite to the course

}

}

}

void printCourseInformation(Tree<Course> courses, String courseNumber) {

(FOR LOOP) all the strored Nodes within the binary search tree “Courses” {

(IF LOOP) the found name of the course equals the found number of the course {

DISPLAY the node's saved details

}

(IF LOOP) the binary search tree “Course” has a left child attached to it {

DISPLAY the node’s information as pre-requisite information

}

(IF LOOP) the binary search tree “Course” has a right child attached to it {

DISPLAY the node’s info as prerequisite information

}

(IF LOOP) the node’s left child exists {

ASSIGN the current node to equal the node’s left child

}

(IF LOOP) the node’s right child exists {

ASSIGN the current node to equal the node’s right child

}

}

}

### Menu Pseudocode

(WHILE LOOP) selection does not equal 4 {

DISPLAY the menu options

(IF LOOP) option 1 is selected (load data structure) {

START the int “numPrerequisiteCourses” function for the Data Structure that is selected (it can be Vector, Hashtable, or Bineary Search Tree)

This function will parse through the CSV file, create the data structure and pass in the courses and pre-requisites

BREAK

}

(IF LOOP) option 2 is selected (Print course list) {

START the “printSampleSchedule” function for the Data Structure that is selected (it can be Vector, Hashtable, or Bineary Search Tree)

DISPLAY the courses’ info and their pre-requisite(s) if any is/are present

This function will go over the data structure that we loaded in the previous option and will print the course’s information such as the name and ID and if the course has pre-requisite(s) it will display those with the course as well

BREAK

}

(IF LOOP) option 3 is selected (print course) {

INPUT the course ID

(IF LOOP) the ID of the course is present in the data structure {

START the “printCourseInformation” function for the Data Structure that is selected (it can be Vector, Hashtable, or Bineary Search Tree)

This function will search for a specific course and will display its information including the pre-requisites

}

ELSE {

DISPLAY a message that states that the course is not present

}

BREAK

}

(IF LOOP) option 4 is selected (EXIT) {

DISPLAY a message stating that the program ended/exited

BREAK

}

}

### Courses in Alphanumeric order

//function to sort courses in alphanumeric order

void alphaNumSort(Vector<Course> courses, indexLow, indexHigh) {

ASSIGN indexMiddle to be a point in the middle of low and high

ASSIGN pivot to equal indexMiddle

(WHILE LOOP) indexLow is equal or smaller than indexHigh {

(WHILE LOOP) vector’s indexLow is less than the assigned pivot point {

ASSIGN indexLow with the new vector position

INCREASE indexLow by one position

}

(WHILE LOOP) vector’s pivot/indexMiddle is less than indexHigh {

REDUCE indexHigh by one position // basically do nothing as the indexHigh is the highest positioned course

}

(IF LOOP) indexLow is higher than indexHigh {

BREAK //the sorting is finished

}

ELSE {

EXCHANGE indexLow and indexHigh positions

}

}

RETURN indexHigh

}

// Print the sorted list to the display

printSortedCourseList {

START the “alphaNumSort” function for the Data Structure

PARSE through the alphanumerically sorted list of courses

DISPLAY each courses’ info: ID and Name

This function will go over the data structure that we loaded and will print the course’s information such as the name and ID

BREAK

}

## Run-time and Memory Evaluation

### Opening the File

|  |  |  |  |
| --- | --- | --- | --- |
| **Line of the Code** | **Line Cost** | **# of Times the Line Executes** | **Total Cost** |
| OPEN file.txt | 1 | 1 | 1 |
| ASSIGN coursesTree = new Tree<Courses>() | 1 | 1 | 1 |
| (FOR LOOP) each line in the file | 1 | n | n |
| ASSIGN listEntries = split(line with comma) | 1 | n | n |
| (IF LOOP) listEntries is less than two | 1 | n | n |
| (IF LOOP) the course has a prerequisite validate if that course exists | 1 | n | n |
| RAISE ERROR message if does not exist | 1 | n | n |
| CREATE a new course with an adequate listEntries | 1 | n | n |
| APPEND the created course to the tree | 1 | n | n |
| **Total Cost** | | | 7n+2 |
| **Runtime** | | | O(n) |

### Vector Analysis

#### Create Vector Data Structure

|  |  |  |  |
| --- | --- | --- | --- |
| **Line of the Code** | **Line Cost** | **# of Times the Line Executes** | **Total Cost** |
| OPEN the file function | 1 | 7n+2 | 7n+2 |
| int numPrerequisiteCourses(Vector<Course> courses, Course c) | 1 | 1 | 1 |
| ASSIGN the variable totalPrerequisites to be equal to all the pre-requisites of “Course c” | 1 | 1 | 1 |
| (FOR LOOP) every pre-requisite located in the variable “totalPrerequisites” | 1 | n | n |
| APPEND the values of pre-requisites into the variable “totalPrerequisites” | 1 | n | n |
| DISPLAY the number that represents all of the saved values in the variable “totalPrerequisites” | 1 | 1 | 1 |
| **Total Cost** | | | 9n+5 |
| **Runtime** | | | O(n) |

### Hashtable Analysis

#### Create Hashtable Data Structure

|  |  |  |  |
| --- | --- | --- | --- |
| **Line of the Code** | **Line Cost** | **# of Times the Line Executes** | **Total Cost** |
| OPEN the file function | 1 | 7n+2 | 7n+2 |
| int numPrerequisiteCourses(Hashtable<Course> courses) | 1 | 1 | 1 |
| ASSIGN the variable “totalPrerequisites” to be equal to the Hash-table of the prerequisites of the courses | 1 | 1 | 1 |
| (FOR LOOP) all the pre-requisites stored in the “totalPrerequisites” variable | 1 | n | n |
| APPEND the values into the new “totalPrerequisites” Hash-table holding the pre-requisites | 1 | n | n |
| APPEND each course with key as ID and value as name of the course | 1 | n | n |
| DISPLAY the number that represents all of the saved values in the variable “totalPrerequisites” | 1 | 1 | 1 |
| **Total Cost** | | | 10n+5 |
| **Runtime** | | | O(n) |

### Binary Search Tree Analysis

#### Create BST Data Structure

|  |  |  |  |
| --- | --- | --- | --- |
| **Line of the Code** | **Line Cost** | **# of Times the Line Executes** | **Total Cost** |
| OPEN the file function | 1 | 7n+2 | 7n+2 |
| int numPrerequisiteCourses(Tree<Course> courses) | 1 | 1 | 1 |
| ASSIGN the variable “totalPrerequisites” to be equal to the left child of the Node | 1 | 1 | 1 |
| ASSIGN the variable “totalPrerequisites” to be equal to the right child of the Node | 1 | 1 | 1 |
| (FOR LOOP) all the pre-requisites stored within the variable “totalPrerequisites” | 1 | n | n |
| APPEND left Node of the pre-requisites to “totalPrerequisites” | 1 | n | n |
| APPEND right Node of the pre-requisites node to “totalPrerequisites” | 1 | n | n |
| DISPLAY the number that represents all of the saved values in the variable “totalPrerequisites” | 1 | 1 | 1 |
| **Total Cost** | | | 10n+6 |
| **Runtime** | | | O(n) |

## Advantages & Disadvantages

### Vector Data Structure

### Hashtable Data Structure

### Binary Search Tree Data Structure

## Recommendation

## References

Andujar D. (2023). CS-300-T3307 DSA: Analysis and Design 23EW3. Word Document provided by the professor (CS 300 Pseudocode Document). Retrieved on 01/29/2023

Novosad D. (2023). CS-300-T3307 DSA: Analysis and Design 23EW3. Word Document from private submission (Milestone Vector Data Structure Pseudocode). Retrieved on 01/29/2023