Project One Pseudocode and Evaluation

Justin Hancock

CS-300 Analysis and Design

// Vector Pseudocode

// DEFINE course

DEFINE class course

Code: String

Name: String

Prerequisites: vector<String>

END CLASS

// Open and Read Data from File, parse through each line and validate information.

BEGIN Read Data Function

DEFINE a vector data structure to hold the courses

INITIALIZE the csv parser using the given path

TRY

DEFINE a vector data structure to hold courseNum

FOR loop to read the rows of the csv file

IF line has less than 2 columns

THEN Throw error "Make sure there are 2 parameters"

END IF

ELSE

Define String courseNum equal to column[0] in row of csv

ADD courseNum to vector<String> courseNum

END ELSE

END FOR

INITIALIZE the csv parser using the given path

FOR loop to read the rows of the csv file

IF prerequisites exist in courseNum

// Creating course object and storing them in a vector

CREATE course object to hold the course code and course name = to first two columns of row

THEN add to vector<Course> courses

ENDIF

END FOR

CATCH csv error

END CATCH

RETURN courses

END Read Data Function

// Search data structure for specific course and print course info and prereqs

BEGIN Search Function

FOR all courses

IF the course is the same as courseNum

PRINT out the course number and name

END IF

FOR each prerequisite of the course object

PRINT the prerequisite course number

END FOR

END FOR

END Search Function

BEGIN Alphanumeric Sort function

DEFINE min

FOR loop to iterate i through the course size

SET min equal to i

FOR loop to iterate j through course size

IF course at j is less than course at i

THEN min = j

END IF

IF min is not i

THEN swap the bids at i and min

END IF

END FOR

END FOR

PRINT sorted list to display

END Alphanumeric Sort function

BEGIN Main function

SET choice to zero

WHILE choice does not equal 4

PRINT 1. Load data structure

PRINT 2. Print course list

PRINT 3. Course

PRINT 4. Exit

BEGIN switch(choice)

Case 1 Read Data function ()

Case 2 Alphanumeric Sort function ()

Case 3 Search function ()

Case 4 Exit

END switch

END while

END Main function

|  |  |  |  |
| --- | --- | --- | --- |
| Vector | Line Cost | Times Executed | Total Cost |
| Define to hold courses | 1 | 1 | 1 |
| Initialize pathing to CSV | 1 | 1 | 1 |
| Define to hold courseNum | 1 | 1 | 1 |
| For loop to read rows of CSV | 1 | n | n |
| Catch CSV error | 1 | 1 | 1 |
| For all courses | 1 | N | N |
| If course number is same as courseNum | 1 | N | N |
| Print course information | 1 | 1 | 1 |
| For each prereq of the course | 1 | N | N |
| Print prereq information | 1 | N | N |
| Total Cost | 5(n) + 5 | | |
| Runtime | O(n) | | |

// Hash Table Pseudocode

// Open and Read Data from File, parse through each line and validate information.

BEGIN Read Data Function

DEFINE a hash table structure to hold the courses

INITIALIZE the csv parser using the given path

TRY

DEFINE a hash table structure to hold courseNum

FOR loop to read the rows of the csv file

IF line has less than 2 columns

THEN Throw error "Make sure there are 2 parameters"

END IF

ELSE

Define String courseNum equal to column[0] in row of csv

ADD courseNum to vector<String> courseNum

END ELSE

END FOR

INITIALIZE the csv parser using the given path

FOR loop to read the rows of the csv file

IF prerequisites exist in courseNum

// Creating course object and storing them in a hash table

CREATE course object to hold the course code and course name equals to first two columns of row

THEN add to hash table<Course> courses

ENDIF

END FOR

CATCH csv error

END CATCH

RETURN courses

END Read Data Function

// Search data structure for specific course and print course info and prereqs

BEGIN Search Function

FOR all courses

IF the course is the same as courseNum

PRINT out the course number and name

END IF

FOR each prerequisite of the course object

PRINT the prerequisite course number

END FOR

END FOR

END Search Function

BEGIN Alphanumeric Sort function

DEFINE hash table = new course table

FOR each hash key

ADD node entries to course table

SORT by course id

PRINT sorted courses

END FOR

END Alphanumeric Sort function ()

BEGIN Main function

SET choice to zero

WHILE choice does not equal 4

PRINT 1. Load data structure

PRINT 2. Print course list

PRINT 3. Course

PRINT 4. Exit

BEGIN switch(choice)

Case 1 Read Data function ()

Case 2 Alphanumeric Sort function ()

Case 3 Search function ()

Case 4 Exit

END switch

END while

END Main function

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Hashtable | Line Cost | Times Executed | | | Total Cost | | |
| Define to hold courses | 1 | 1 | | | 1 | | |
| Initialize pathing to CSV | 1 | 1 | | | 1 | | |
| Define to hold courseNum | 1 | 1 | | | 1 | | |
| For loop to read rows of CSV | 1 | n | | | n | | |
| Catch CSV error | 1 | 1 | | | 1 | | |
| For all courses | 1 | N | | | N | | |
| If course number is same as courseNum | 1 | N | N | | |
| Print course information | 1 | 1 | 1 | | |
| For each prereq of the course | 1 | N | | N | | |
| Print prereq information | 1 | N | | N | | |
| Total Cost | 5(n) + 5 | | | | |
| Runtime | O(n) | | | | |

// Binary Search Tree Pseudocode

// Open and Read Data from File, parse through each line and validate information.

BEGIN Read Data Function

DEFINE a Binary Search Tree structure to hold the courses

INITIALIZE the csv parser using the given path

TRY

DEFINE a Binary Search Tree structure to hold courseNum

FOR loop to read the rows of the csv file

IF line has less than 2 columns

THEN Throw error "Make sure there are 2 parameters"

END IF

ELSE

Define String courseNum equal to column[0] in row of csv

ADD courseNum to vector<String> courseNum

END ELSE

END FOR

INITIALIZE the csv parser using the given path

FOR loop to read the rows of the csv file

IF prerequisites exist in courseNum

// Creating course object and storing them in a tree

CREATE course object to hold the course code and course name equals to first two columns of row

THEN add to hash table<Course> courses

ENDIF

END FOR

CATCH csv error

END CATCH

RETURN courses

END Read Data Function

// Search data structure for specific course and print course info and prereqs

BEGIN Search Function

FOR all courses

IF the course is the same as courseNum

PRINT out the course number and name

END IF

FOR each prerequisite of the course object

PRINT the prerequisite course number

END FOR

END FOR

END Search Function

BEGIN Alphanumeric Sort function ()

IF node does not equal nullptr

THEN call inOrder recursively with left node

PRINT course information

THEN call inOrder recursively with right node

END IF

END Alphanumeric Sort function ()

BEGIN Main function

SET choice to zero

WHILE choice does not equal 4

PRINT 1. Load data structure

PRINT 2. Print course list

PRINT 3. Course

PRINT 4. Exit

BEGIN switch(choice)

Case 1 Read Data function ()

Case 2 Alphanumeric Sort function ()

Case 3 Search function ()

Case 4 Exit

END switch

END while

END Main function

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Binary Search Tree | Line Cost | Times Executed | Total Cost | |
| Define to hold courses | 1 | 1 | 1 | |
| Initialize pathing to CSV | 1 | 1 | 1 | |
| Define to hold courseNum | 1 | 1 | 1 | |
| For loop to read rows of CSV | 1 | n | n | |
| Catch CSV error | 1 | 1 | 1 | |
| For all courses | 1 | N | N | |
| If course number is same as courseNum | 1 | N | | N |
| Print course information | 1 | 1 | | 1 |
| For each prereq of the course | 1 | N | | N |
| Print prereq information | 1 | N | | N |
| Total Cost | 5(n) + 5 | | | |
| Runtime | O(n) | | | |

Each one of the data structures has its own advantages and disadvantages. For the requirements of the program the way I have written the code, the runtime is the same for each example. Although, I am not extremely confident in my calculations on coming up with the runtime for each section of the program. I did some further research on bigocheatsheet.com and found a few pieces that helped me better understand the differences in run times between the data structures themselves. According to the chart listed on the website, a hash table would have the fastest worst time complexity at O(1), but it does not have the ability to provide an ordered list of the data. So, I’m not sure that a hash table would be best used for this program. A binary search tree would have the next fastest worst time complexity at O(log(n)), and it would have the ability to create an ordered list of the data. A doubly linked list (vector) would have the worst time complexity at O(n), which is still a fair speed of the program to run. A doubly linked list does have some features like insertion and deletion that run at O(1), but I think the structure would be better used for organizing different types of data.

I would recommend using a binary search tree for this program. It is easy to access values from smallest to largest using the recursive calls on the needed sort. The creator can change the order of the root for a binary search tree and customize the tree in multiple ways which makes the tree easy to populate with values. Since all the data structures have similar time complexities, I think the easiest way to create this program is using a binary search tree.

**Citation**

*Big-O Algorithm Complexity Cheat Sheet (Know Thy Complexities!) @ericdrowell*. (n.d.). https://www.bigocheatsheet.com/