Jessica Ayer

CS 300: Project 1

04/09/2022

**Pseudocode**

**//Vector**

vector<courses> loadCourses(string, csvPath){

vector <course> courses

CALL parser

FOR rows in CSV file loop to read

CREATE data structure

courseNumber[i][0]

courseName[i][1]

FOR rest of Line

PUSH\_BACK prerequisites vector

PUSH\_BACK course

RETURN course

}

createCourseObject(){

INTIATE data structure

BOOL prerequisite = False

INPUT courseNumber

INPUT courseName

INPUT prerequisite T or F

IF input “T” prerequisite = True

Input coursePrerequisite

PUSH\_BACK prerequiste

INPUT prerequisite T or F

PUSH\_BACK course

RETURN course

}

printCourseInformation(Course course){

FOR all courses

IF course is the same as courseNumber

OUTPUT courseNumber

OUTPUT courseName

FOR each prerquisite

OUTPUT prerequisite

}

alphanumericPrintAll(){

FOR all elements in vector

FIND smallest remaining element

FOR min+1

IF min+1 is less then min

min = min +1

IF min not i

Swap i and min

FOR all elements in vector

printCourseInfomration(course[i])

}

**//hash table**

loadCourses(string cvsPath, HashTable\* hashTable){

CALL parser

FOR rows in CSV file loop to read

courseNumber [i][0]

courseName [i][1]

FOR rest of Line

PUSH\_BACK prerequisites vector

Hash Table-> INSERT course

//Insert

CREATE key

RETRIEVE node using key

IF Node is null

NEW node is this course and key

INSERT node begin + Key, new node

ELSE

IF node key equals UNIT\_MAX

node->key = key

node-> course = course

node->next = null

ELSE

WHILE node->next is not null

node = node->next

node->next = new node

//end Insert

}

createCourseObject(){

INTIATE data structure

BOOL prerequisite = False

INPUT courseNumber

INPUT courseName

INPUT prerequisite T or F

IF input “T” prerequisite = True

Input coursePrerequisit

INPUT prerequisite T or F

HashTable->INSERT course

}

printCourse(Course course){

FOR all course

IF the course is the same as courseNumber

OUTPUT courseNumber

OUTPUT courseName

For each prerequisite of hashTable[course]

OUTPUT prerequisite

OUTPUT courseName

RETURN

}

alphanumericPrintAll(){

FOR each node

IF the key does not equal UINT\_MAX

OUTPUT courseNumber

OUTPUT courseName

IF course has prerequisite

FOR each prerequisite

OUTPUT prerequisite

WHILE node does not equal a null pointer

OUTPUT courseNumber

OUTPUT courseName

IF course has prerequisite

For each prerequisite

OUTPUT prerequisit

Node = node->next

}

**//Binary Search Tree**

loadCourses(string vscPath, BinarySearchTree\* bst){

CALL parser

FOR rows in CSV file loop to read

courseNumber [i][0]

courseName [i][1]

FOR rest of Line

PUSH\_BACK prerequisites vector

Binary Search Tree -> INSERT course

//Insert

IF root node is null

Root node = new node course

ELSE

Add THIS node root and course

//Add

IF node is larger then add to left

IF no left node

This node becomes left node

ELSE

Recurse down the left node

ELSE

IF no right node

This node becomes right node

ELSE

Recurse down the right node

//end ADD

//end Insert

}

createCourseObject(Node\*node Course course){

BOOL prerequisite = False

INPUT courseNumber

INPUT courseName

INPUT prerequisite T of F

IF input “T” prerequisite = True

Input coursePrerequisite

PUSH\_BACK prerequisite

INPUT prerequisite T of F

Binary Search Tree->INSERT course

}

printCourse(Course course){

FOR all nodes

IF the course s the same as courseNumber

OUTPUT courseName

OUTPUT courseNumber

IF prerequisite //left node

OUTPUT prerequisite

IF prerequisite //right node

OUTPUT prerequiste

ELSE IF

IF course is smaller than courseNumber

Traverse left

ELSE

Traverse right

RETURN

}

alphanumericPrintAll(Node\* node){

IF node does is not null

InOrder left

OUTPUT course number and course name

IF has left node //prerequisite

OUTPUT prerequisite

IF has right node //prerequisite

OUTPUT prerequiste

InOrder right

}

**Menu:**

int choice = 0

WHILE choice does not equal 9

OUTPUT menu title

OUTPUT “1. Load Courses”

OUTPUT “2. Print Course List”

OUTPUT “3. Print Course”

OUTPUT ”9. Exit”

SWITCH(choice){

CASE 1:

CALL method loadCourse

CASE 2:

CALL method alphanumericPrintAll

CASE 3:

CALL method printCourse

EXIT

**Time Complexities**

**Vector:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Code: Reading a file** | **Line Cost** | **# Times Executed** | **Total Cost** |
| Vector<course>courses | 1 | 1 | 1 |
| CALL parser | 1 | 1 | 1 |
| FOR rose in CSV file loop to read | 1 | n | n |
| CREATE data structure | 1 | n | n |
| courseNumber[i][0] | 1 | n | n |
| courseName[i][1] | 1 | n | n |
| FOR rest of Line | 1 | n | n |
| PUSH\_BACK prerequisites vector | 1 | n | n |
| PUSH\_BACK course | 1 | n | n |
| RETURN course | 1 | 1 | 1 |
| **Total Cost** | | | 7n+3 |
| **Runtime** | | | O(n) |

|  |  |  |  |
| --- | --- | --- | --- |
| **Code: Creating an Object** | **Line Cost** | **# Times Executed** | **Total Cost** |
| INITATE data structure | 1 | 1 | 1 |
| BOOL prerequisite = FALSE | 1 | 1 | 1 |
| INPUT courseNumber | 1 | 1 | 1 |
| INPUT courseName | 1 | 1 | 1 |
| INPUT prerequiste T or F | 1 | 1 | 1 |
| IF input “T” prerequiste = True | 1 | n | n |
| INPUT coursePrerequiste | 1 | 1 | 1 |
| PUST\_BACK prerequiste | 1 | 1 | 1 |
| INPUT prerequiste T or F | 1 | 1 | 1 |
| **Total Cost** | | | n+8 |
| **Runtime** | | | O(n) |

**Hash Table:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Code: Load courses** | **Line Cost** | **# Times Executed** | **Total Cost** |
| CALL parser | 1 | 1 | 1 |
| FOR rows in CSV file loop to read | 1 | n | n |
| courseNumber[i][0] | 1 | 1 | 1 |
| courseName[i][1] | 1 | 1 | 1 |
| FOR rest of Line | 1 | n | n |
| PUSH\_BACK prerequisites vector | 1 | 1 | 1 |
| Hash Table ->Insert course | 1 | 1 | 1 |
| CREATE key | 1 | 1 | 1 |
| RETRIEVE node using key | 1 | 1 | 1 |
| IF node is null | 1 | n | n |
| NEW node is this course and key | 1 | 1 | 1 |
| INSERT node begin + Key new node | 1 | 1 | 1 |
| **Total Cost** | | | 3n+9 |
| **Runtime** | | | O(n) |

|  |  |  |  |
| --- | --- | --- | --- |
| **Code: Create Course Object** | **Line Cost** | **# Times Executed** | **Total Cost** |
| INITIATE data structure | 1 | 1 | 1 |
| BOOL prerequisite = False | 1 | 1 | 1 |
| INPUT courseNumber | 1 | 1 | 1 |
| INPUT courseName | 1 | 1 | 1 |
| IF input “T” prerequisite = True | 1 | n | n |
| INPUT coursePrerequisite | 1 | 1 | 1 |
| PUSH\_Back prerequisite vector | 1 | 1 | 1 |
| INPUT another prerequisite or F | 1 | 1 | 1 |
| HashTable->INSERT course | 1 | 1 | 1 |
| **Total Cost** | | | n+8 |
| **Runtime** | | | O(n) |

**Binary Search Tree:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Code: Load Courses** | **Line Cost** | **# Times Executed** | **Total Cost** |
| CALL parser | 1 | 1 | 1 |
| FOR rows in CSV file loop to read | 1 | n | n |
| courseNumber[i][0] | 1 | 1 | 1 |
| courseName[i][1] | 1 | 1 | 1 |
| FOR rest of line | 1 | n | n |
| PUSH\_BACK prerequisites vector | 1 | 1 | 1 |
| BinarySearchTree->INSERT course | 1 | 1 | 1 |
| IF root node is null | 1 | n | n |
| ROOT node = new node course | 1 | n | n |
| **Total Cost** | | | 4n+5 |
| **Runtime** | | | O(n) |

|  |  |  |  |
| --- | --- | --- | --- |
| **Code: Create Course Object** | **Line Cost** | **# Times Executed** | **Total Cost** |
| INITIATE data structure | 1 | 1 | 1 |
| BOOL prerequisite = False | 1 | 1 | 1 |
| INPUT courseNumber | 1 | 1 | 1 |
| INPUT courseName | 1 | 1 | 1 |
| IF input “T” prerequisite = True | 1 | n | n |
| INPUT coursePrerequisite | 1 | 1 | 1 |
| PUSH\_Back prerequisite vector | 1 | 1 | 1 |
| INPUT another prerequisite or F | 1 | 1 | 1 |
| BinarySearchTree->INSERT course | 1 | 1 | 1 |
| **Total Cost** | | | n+8 |
| **Runtime** | | | O(n) |

**Evaluation:**

For this project it appears that all three data structures, vector, hash table and binary search tree all have the same worst-case runtime. Representing the number of elements being loaded into the data structure with n, all three data structures have a worse case runtime of O(n) for loading courses or creating new course objects. If we were to solely base this project off the worst case run time for these two methods it would not matter which data structure we used however, in hopes that the application will not require worst case runtime for every instance of use, we can also take best case or average runtimes into consideration as well and some of the other methods that will be needed such as printing and searching. According the Rowell’s website *Big O cheat sheet*, with help from contributors the average runtime for insertions into a is O(1) as is a Hash Table and a binary Search Tree is O(log(n)). There is not a significant difference in time complexities however, if you look at the search method which will be used in this application as well, we start to see a larger difference in the runtimes. vectors have a time complexity of O(n), hash tables of O(1), and binary search trees of O(log(n)) (Rowell). Looking at a time complexity graph, the runtime of vectors becomes significantly longer as the number of elements increase (Vahid, 2019).

**Recommendation:**

Due to the time complexities for various methods, I would eliminate the option of using vectors as the primary data structure for this application. The data structure needs the flexibility to add and delete an unknown number of courses that have an unknown number of prerequisites. It also needs to be able to sort the courses alphanumerically. I would personally choose to use a binary search tree for this application. A binary search tree makes it easier to search for specific elements and to sort the courses in various ways and can perform all of the required tasks with a minimal runtime.

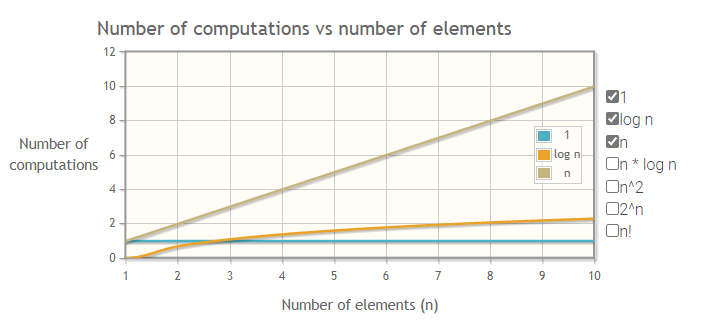
****

Image from *Data Structures and Algorithms* (Vahid, 2019)

**References:**

Rowell, Eric. *Know the Complexities.* Big-O Cheat Sheet.  [https://www.bigocheatsheet.com/#:~:text=Common%20Data%20Structure%20Operations%20%20%20%20Data,%CE%98%20%28log%20%28n%29%29%20%2012%20more%20rows%20gorithm Complexity Cheat Sheet (Know Thy Complexities!) @ericdrowell (bigocheatsheet.com)](https://www.bigocheatsheet.com/#:~:text=Common%20Data%20Structure%20Operations%20%20%20%20Data,%CE%98%20%28log%20%28n%29%29%20%2012%20more%20rows%20)

Vahid, F., Lysecky, S. Siu, R. (February 2019) *Data Structures and Algorithms.* Zyante Inc. https://learn.zybooks.com/zybook/SNHUCS300v1