**Jeremy Depenhart**

**Project One**

**CS 300: DSA: Analysis and Design**

**Southern New Hampshire University**

**6/9/2023**

**Pseudocode:**

**Check File: //All**

Start

Initialize bool Prerequisite Exists

Open file with Parser

If File wont open

Print error

Exit program

For all row in File

Read row

Strip line into parameters at blank space

Set first parameter as course number

If second parameter is null

Print error

Exit program

Else

Set second parameter as course name

If more than two parameters exist

Loop until no additional parameters

Set parameter a prerequisite

While Not end of file

Read line

Strip line into parameters at blank space

Set first parameter as course number

If course number equals prerequisite

Set Prerequisite Exists to true

Else

Set Prerequisite Exists to false

If Prerequisite Exists is false

Print error

Exit program

Close file

END

**Vector Create course object: // Vector**

Start

Initialize file vector

Initialize Course ID

Initialize Title

Initialize Prerequisite

Open File

For all line in file

Create file vector elements

Set Course ID as First parameter

Set Title as Second parameter.

If there are three or more parameters

Set Prerequisite as other parameters in line

Close File

End

**Hashtable Create course object: // Hashtable**

Start

Initialize File Stream fileStream

Initialize Course Object currentCourse

Open File

While not end of file

Read Line

strip line into parameters at blank space

fileStream take in first parameter as courseNumber

Set currentCourse.number as courseNumber

fileStream take in second parameter as courseName

Set currentCourse.name as courseName

If more than two parameters exist

Set currentCourse.prerequisites as additional parameters

currentCourse points to next line

Close File

End

**Tree Create course object:**

**Create course object: //Tree**

Start

Initialize Parser file as file path

For all row in file

Initialize Course object

Read row

Strip line into parameters at blank space

Set Course Number to first parameter

Set Course Title to second parameter

If more than two parameters exist

Set Course Prerequisite to additional parameters

Call Insert Course function with Course object

End

**Insert Course: //Tree**

Start

Take in Course object

If first node of binary tree is null ptr

Set first node to new Node Course object

Else

Call Add Node function with first node and Course object

End

**Add Node: //Tree**

Start

Take in node and Course object

If Course number is larger than node Course number

If no node exists to the left

Set left node to new node Course object

Else

Recurse Add Node function with left node and Course object

Else

If no node exists to the right

Set right node to new node Course object

Else

Recurse Add Node function with right node and Course object

End

**Print course information: // All**

Start

Take in Course Number

Initialize bool Course Found as false

For all elements in data structure

If This Course Number equals element’s Course Number

Set Course Found as true

Print Course information

Return

If Course Found is false

Print Course Not Found

End

**Print courses Alphanumerically // All**

Start

Take in data structure

Initialize temporary data structure

Set temporary data structure as data structure

Sort temporary data structure beginning to end //use built in sort function

For all elements in temporary data structure

Print Element information

End

**Menu // All**

Start

Initialize integer choice as 0

Initialize Course number

While choice does not equal 4

Print menu options

Input choice

Switch case for choice

Default

Print Error statement

Case 1

Call Create Course Object function // Different based on data structure

Print Success statement

Case 2

Call Print Courses Alphanumerically function

Case 3

Input Couse Number

Call Print course information function with Couse Number

Case 4

Break Loop

Print Goodbye

End

**Evaluation**

**Run Time Analysis**

| **Vector** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **Initialize file vector** | 1 | 1 | 1 |
| **Open File** | 1 | 1 | 1 |
| **For all line in file** | 1 | n | n |
| **Set Course ID as First parameter** | 1 | 1 | 1 |
| **Set Title as Second parameter** | 1 | 1 | 1 |
| **If there are three or more parameters** | 1 | n | n |
| **Set Prerequisite as other parameters in line** | 1 | 1 | 1 |
| **Close File** | 1 | 1 | 1 |
| **Total Cost** | | | 2n + 6 |
| **Runtime** | | | O(n) |

| **Hash Table** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **Initialize File Stream fileStream** | 1 | 1 | 1 |
| **Initialize Course Object currentCourse** | 1 | 1 | 1 |
| **Open File** | 1 | 1 | 1 |
| **While not end of file** | 1 | n | n |
| **strip line into parameters at blank space** | 1 | 1 | 1 |
| **fileStream take in first parameter as courseNumber** | 1 | 1 | 1 |
| **Set currentCourse.number as courseNumber** | 1 | 1 | 1 |
| **fileStream take in second parameter as courseName** | 1 | 1 | 1 |
| **Set currentCourse.name as courseName** | 1 | 1 | 1 |
| **If more than two parameters exist** | 1 | n | n |
| **Set currentCourse.prerequisites as additional parameters** | 1 | 1 | 1 |
| **currentCourse points to next line** | 1 | 1 | 1 |
| **Close File** | 1 | 1 | 1 |
| **Total Cost** | | | 2n + 11 |
| **Runtime** | | | O(n) |

| **Tree** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **Initialize Parser file as file path** | 1 | 1 | 1 |
| **For all rows in file** | 1 | n | n |
| **Read row** | 1 | 1 | 1 |
| **Strip line into parameters at blank space** | 1 | 1 | 1 |
| **Set Course Number to first parameter** | 1 | 1 | 1 |
| **Set Course Title to second parameter** | 1 | 1 | 1 |
| **If more than two parameters exist** | 1 | n | n |
| **Set Course Prerequisite to additional parameters** | 1 | 1 | 1 |
| **Call Insert Course function with Course object** | Log n | Log n | Log n |
| **Total Cost** | | | Log + 2n + 6 |
| **Runtime** | | | O(Log N) |

**Consideration and Recommendation:**

Each data structure has its own advantages and disadvantages. One’s choice in data structure ultimately depends on the data type and the amount of data they wish to store. In this specific scenario, the required specifications are minimal. There are not many courses listed for the computer science program, at least not enough to where runtime would be that much of an issue. However, if the university wanted to expand the course roster or add more functionality to this program, then the choice of data structure would matter more.

Each of these three structures are well organized and will aid in the code’s legibility. They also allow for easy insertion, searching, and removal of data nodes. The main factor distinguishing each of them is runtime. As seen above, both the vector and hash table data structures end on an O(n) runtime cost after the removal of constants. The tree data structure is the only one which ends on a logarithmic n.

For this specific project, my recommendation would be to use either the vector or hash table data structure. A tree structure costs too much for the purposes of this program and could prove complicated if the university wished to expand. Between a vector and hash table, I believe the better choice is a vector. This ultimately is decided by the project’s specifications. A vector is much simpler to create compared to a hash table, and this project’s specifications do not call for complexity. If the program’s functionality were to expand, then perhaps a hash table would be better suited for it.