Steven Rodas

CS-300

Professor Andujar

## **Project One**

Included in this document:

* Pseudocode for menu
* Pseudocode for printing out list of courses
* Runtime analysis
* Evaluation
* Recommendation

**Pseudocode for menu -**

Set up a loop that continues indefinitely

Inside the loop, display the menu options to the user:

Load Data Structure

Print Course List

Print Course

Exit

Prompt the user to enter their choice

If the user's choice is "Load Data Structure":

Run the code to load the data into the data structure

If the user's choice is "Print Course List":

Run the code to print an alphabetically sorted list of courses

If the user's choice is "Print Course":

Run the code to print the title and prerequisites of a specific course

If the user's choice is "Exit":

Exit();

If the user's choice is none of the above:

Display an error message to the user

Go back to the beginning of the loop

**Pseudocode for printing out list of course -**

def print\_course\_list():

# sort the list of courses

sorted\_courses = sort(courses)

# print the sorted list of courses

for course in sorted\_courses:

print(courses)

**Runtime Analysis:**

To compute runtime analysis and calculate the total cost and runtime of your piece of code, we must assume there are “n” courses stored in the data structure and assume that the cost for a line that executes is 1 unless it is calling another function or itself.

**Vector**

| **Execution** | **Line Cost** | **Total times executed** | **Total Cost** |
| --- | --- | --- | --- |
| Open file | 1 | 1 | 1 |
| Create empty vector | 1 | n | n |
| Create empty dictionary | 1 | n | n |
| For each line in file | 1 | n | n |
| Split line by commas | 1 | n | n |
| If there are fewer than two fields | 1 | n | n |
| Print error message | 1 | n | n |
| Skip to next line | 1 | n | n |
| Get the first field | 1 | n | n |
| Get the second field | 1 | n | n |
| If course # is already in course data | 1 | n | n |
| Print error | 1 | n | n |
| Skip to next line | 1 | n | n |
| Add course number and course title to course data | 1 | n | n |
| Set prerequisites to all fields after 2nd field if there are more than 2 fields | 1 | n | n |
| For each prereq in prerequisites | 1 | n | n |
| If the prereq is NOT in course data | 1 | n | n |
| Print error message | 1 | n | n |
| Skip to next line | 1 | n | n |
| Create new course object | 1 | n | n |
| Append course object to end of courses vector | 1 | n | n |
| For each course in courses | 1 | n | n |
| Print course number and title | 1 | n | n |
| End project | 1 | n | n |
|  |  | **Total Cost:** | 23n+1 |
|  |  | **Runtime:** | O(n) |

**Hash Table**

| **Execution** | **Line Cost** | **Total times executed** | **Total Cost** |
| --- | --- | --- | --- |
| Open file | 1 | 1 | 1 |
| Create empty vector | 1 | n | n |
| Create empty dictionary | 1 | n | n |
| For each line in file | 1 | n | n |
| Split line by commas | 1 | n | n |
| If there are fewer than two fields | 1 | n | n |
| Print error message | 1 | n | n |
| Skip to next line | 1 | n | n |
| Get the first field | 1 | n | n |
| Get the second field | 1 | n | n |
| If course # is already in course data | 1 | n | n |
| Print error | 1 | n | n |
| Skip to next line | 1 | n | n |
| Add course number and course title to course data | 1 | n | n |
| Set prerequisites to all fields after 2nd field if there are more than 2 fields | 1 | n | n |
| For each prereq in prerequisites | 1 | n | n |
| For each prereq in prerequisites | 1 | n | n |
| If the prereq is NOT in course data | 1 | n | n |
| Print error message | 1 | n | n |
| Skip to next line | 1 | n | n |
| Create new course object | 1 | n | n |
| Append course object to end of courses hash table by using the courses key | 1 | n | n |
| Close file | 1 | n | n |
| For each key in the hash table | 1 | n | n |
| Get the course object using the key | 1 | n | n |
| Print course number and title | 1 | n | n |
| End program | 1 | n | n |
|  |  | **Total Cost:** | 26n+1 |
|  |  | **Runtime:** | O(n) |

**Binary Tree**

| **Execution** | **Line Cost** | **Total times executed** | **Total Cost** |
| --- | --- | --- | --- |
| Open file | 1 | 1 | 1 |
| Create empty vector | 1 | n | n |
| Create empty dictionary | 1 | n | n |
| For each line in file | 1 | n | n |
| Split line by commas | 1 | n | n |
| If there are fewer than two fields | 1 | n | n |
| Print error message | 1 | n | n |
| Skip to next line | 1 | n | n |
| Get the first field | 1 | n | n |
| Get the second field | 1 | n | n |
| If course # is already in course data | 1 | n | n |
| Print error | 1 | n | n |
| Skip to next line | 1 | n | n |
| Add course number and course title to course data | 1 | n | n |
| Set prerequisites to all fields after 2nd field if there are more than 2 fields | 1 | n | n |
| For each prereq in prerequisites | 1 | n | n |
| For each prereq in prerequisites | 1 | n | n |
| If the prereq is NOT in course data | 1 | n | n |
| Print error message | 1 | n | n |
| Skip to next line | 1 | n | n |
| Create new course object | 1 | n | n |
| Append course object to end of courses binary tree using course number as key | 1 | n | n |
| Close file | 1 | n | n |
| Call inorder traversal function | 1 | n | n |
| For each course in traversal | 1 | n | n |
| Print course number and title | 1 | n | n |
| End program | 1 | n | n |
|  |  | **Total Cost:** | 26n+1 |
|  |  | **Runtime:** | O(n) |

**Evaluation**

A vector, otherwise known as an array, is a data structure that stores a collection of items and allows them to be accessed by their given indices. One advantage of a vector is that it allows for faster access to individual elements because of its index being used to directly access the element. Vectors can also accommodate for new information as they can easily be resized. The biggest disadvantage to the vector data structure, in my opinion, is the fact that appending or removing elements of the vector data structure could be inefficient and increase runtime. When appending or removing elements, vectors have to shift every other existing element which would in turn develop into a slower working program.

A hash table is a data structure that stores a collection of items in an array but uses a built-in hash function that maps individual items to specified indices in the array itself. The best advantage of a hash table is that it allows for fast insertion, removal, and access of individual elements. The biggest disadvantage I found to hash tables is that not all hash functions are created equal and you must ensure you have an efficient hash function that will work. Because of this, you risk a slower working program if you do not implement the hash function correctly.

A binary tree is a data structure that consists of “nodes” and a “root”. Each individual node can have up to two “children”. An advantage of using a binary tree data structure is that is allows for quick and efficient searching within the tree itself. With binary trees, you are able to traverse through the tree to search for a specific element. Binary trees are also extremely viable to be modified, such as insertion and removal of elements, since nodes can be added or removed easily by traversal. A disadvantage of the binary tree data structure that I have found is that the performance and efficiency of a binary tree can hinder if that tree is not developed correctly or becomes inefficient with time and elements added.

**Recommendation**

I handed in this document earlier and realized I did not update my recommendation portion. This is an updated version.

Based on my runtime analysis and evaluation, I have decided that I wanted to develop my program using the Binary Tree data structure. I chose this option because out of all the data structures we have learned in this course, I felt that Binary Trees were a bit confusing for me so I decided to work on a Binary Tree so I can finally grasp the concept of it. On top of this, binary trees are extremely interesting and a viable way to collect, store and modify data. The use of nodes in Binary Trees gives developers great visualization of how their data structure is formatted.