**Vector:**

START

Function loadCourses

Pass in csvPath

Define a vector to hold courses.

OPEN file.

IF file is found:

WHILE it is not the end of the file:

Get each line.

IF Line does not contain course number and name:

Skip to next line.

ELSE IF line contains prerequisite course(s):

Search file for existing course number.

IF course number does not exist in file:

Show error.

End function.

Function createCourse

OPEN file.

IF file did not open:

Print error.

Return.

WHILE there are lines in the file:

Create new course object in vector.

Use stringstream for lines.

Use getline to assign courseNumber to course object, ending at the comma.

User getline to assign the name to course object, ending at next comma.

DO:

Create a temp value.

Use getLine to get prerequisite and assign to temp value.

IF temp value exists in data as a course:

Add to prerequisites of course object.

Delete temp.

WHILE there are prerequisites to sort.

CLOSE file.

End function.

|  |  |  |  |
| --- | --- | --- | --- |
| **Code** | **Line Cost** | **# Times Executed** | **Total Cost** |
| Open file | 1 | 1 | 1 |
| If file doesn’t open | 1 | 1 | 1 |
| Print error | 1 | 1 | 1 |
| Return | 1 | 1 | 1 |
| While there are lines in the file | 1 | n | n |
| Create new course object in vector | 1 | 1 | 1 |
| Stringstream to break up line | 1 | 1 | 1 |
| Use getline to assign courseNumber | 1 | 1 | 1 |
| Use getline to assign course name | 1 | 1 | 1 |
| While there are prerequisites to sort | 1 | n | n |
| Create a temp value | 1 | 1 | 1 |
| Use getline, assign prereq. to temp value | 1 | 1 | 1 |
| IF temp value exists in data as a course | 1 | n | n |
| Add prerequisite to course object | 1 | 1 | 1 |
| Delete temp value | 1 | 1 | 1 |
| Close file | 1 | 1 | 1 |
|  |  | **Total Cost** | 3n + 14 |
|  |  | **Runtime** | O(n) |

**HashMap:**

START

Function loadCourses

Pass in csvPath

Define a vector to hold courses.

OPEN file.

IF file is found:

WHILE it is not the end of the file:

Get each line.

IF Line does not contain course number and name:

Skip to next line.

ELSE IF line contains prerequisite course(s):

Search file for existing course number.

IF course number does not exist in file:

Show error.

End function.

Function createCourse

OPEN file.

IF file did not open:

Print error.

Return.

WHILE there are lines in the file:

Create new course object in vector.

Use stringstream for lines.

Use getline to assign courseNumber to course object node, ending at the comma.

User getline to assign the name to course object node, ending at next comma.

DO:

Create a temp value.

Use getLine to get prerequiste and assign to temp value.

IF temp value exists in data as a course:

Add to prerequistes of course object node.

Delete temp.

WHILE there are prerequisites to sort.

Add course object to bucket

CLOSE file.

End function.

|  |  |  |  |
| --- | --- | --- | --- |
| Code | Line Cost | # Times Executed | Total Cost |
| Open file | 1 | 1 | 1 |
| If file doesn’t open | 1 | 1 | 1 |
| Print error | 1 | 1 | 1 |
| Return | 1 | 1 | 1 |
| While there are lines in the file | 1 | n | n |
| Create new course object | 1 | 1 | 1 |
| Stringstream to break up line | 1 | 1 | 1 |
| Use getline to assign courseNumber | 1 | 1 | 1 |
| Use getline to assign course name | 1 | 1 | 1 |
| While there are prerequisites to sort | 1 | n | n |
| Create a temp value | 1 | 1 | 1 |
| Use getline, assign prereq. to temp value | 1 | 1 | 1 |
| IF temp value exists in data as a course | 1 | n | n |
| Add prerequisite to course object | 1 | 1 | 1 |
| Delete temp value | 1 | 1 | 1 |
| Add course object to bucket | 1 | 1 | 1 |
| Close file | 1 | 1 | 1 |
|  |  | **Total Cost** | 3n + 15 |
|  |  | **Runtime** | O(n) |

**Binary Search Tree:**

START

Function loadCourses

Pass in csvPath

Define a tree to hold courses.

OPEN file.

IF file is found:

WHILE it is not the end of the file:

Get each line.

IF Line does not contain course number and name:

Skip to next line.

ELSE IF line contains prerequisite course(s):

Search file for existing course number.

IF course number does not exist in file:

Show error.

End function.

Function createCourse

OPEN file.

IF file did not open:

Return error

WHILE there are lines in the file:

Create new course object.

Use stringstream getline to assign to course info to course object.

WHILE the course has prerequisites:

Create a temp value.

Use getline to collect prerequisite, assign to temp value.

IF temp value exists as another courseNumber:

Add to prerequisites of course object.

Delete temp value.

IF the tree’s root is null:

Assign root with new Node, pass in course object.

ELSE:

Call to addNode function, pass in root and course object.

End function.

Function addNode

Pass in node and course object

IF node’s courseNumber is > the object’s courseNumber:

IF node->left is null:

Assign node->left with new node, pass in course object.

ELSE:

Call addNode function, pass in node->left and course object.

ELSE:

IF node->right is null:

Assign node->right with new node, pass in course object.

ELSE:

Call addNode function, pass in node->right and course object.

End function.

|  |  |  |  |
| --- | --- | --- | --- |
| Code | Line Cost | # Times Executed | Total Cost |
| Define a tree to hold courses | 1 | 1 | 1 |
| Open file | 1 | 1 | 1 |
| If file isn’t open | 1 | 1 | 1 |
| Return error | 1 | 1 | 1 |
| While there are lines in the file | 1 | n | n |
| Create new course object | 1 | 1 | 1 |
| Use stringstream to break up line | 1 | 1 | 1 |
| Use getline to collect course number | 1 | 1 | 1 |
| Use getline to collect course name | 1 | 1 | 1 |
| While there are prerequisites | 1 | n | n |
| Create temp value | 1 | 1 | 1 |
| IF temp value exists as a course number | 1 | n | n |
| Add temp value as prerequisite of course object | 1 | 1 | 1 |
| Delete temp value | 1 | 1 | 1 |
| IF tree’s root is null | 1 | n | n |
| Assign root with new node, pass in course object | 1 | 1 | 1 |
| Else | 1 | 1 | 1 |
| Call to addNode, pass in root and course object | O(1) | 1 | O(1) |
|  |  | **Total Cost** | O(1) + 4n + 16 |
|  |  | **Runtime** | O(n) |

**Pseudocode for Menu and Print List**

Function printCourseInformation

Pass in course object

Create tracking field, i

WHILE i < the size of the data structure:

IF the node’s course matches the courseNumber:

print out the course information.

FOR each prerequisite:

Print prerequisite course information.

End function.

Function Menu

WHILE the user does not select quit:

Display menu options.

Collect user input.

Case for Loading Data Structure:

Call loadCourses function, pass in csvPath.

Break.

Case for Printing Course List:

Call printCourseList function, pass in vector.

Break.

Case for Print Course:

Request course number.

Get user input.

Call function findCourse, pass in course object and user input.

Break.

Return.

End function

**Advantages and Disadvantages:**

Vectors have the advantage that they provide the user with random access to the elements inside them and can quickly allocate memory as needed through resizing. Vectors also have the benefit of being able to use the index as an expression to quickly look up and access items in the structure. In conjunction with other vector functions, it can be used to easily store ordered data. However, a disadvantage of vectors is that it can be problematic depending on the project as you are not able to insert data at the beginning or middle of the structure.

A hash table works by storing unordered items into an array or vector, mapping them to an index based on a given key. Hash tables have the advantage of being able to look up, insert, and delete data at a very efficient rate. However, hash tables are not naturally ordered and have the risk of collisions. There are methods that can be put in place to help avoid collisions, such as re-hashing, but this impacts the overall efficiency of the data structure.

A binary search tree does have a greater time complexity than a Hash Table or Vector, however it has some advantages over the two. One benefit of using a Binary Search Tree is that you do not need to know the approximate size of your data beforehand. There is no need for resizing or rehashing, which can affect the runtime. Whenever new space is needed for data, you can just add a new node to the tree. Using this data structure also provides sorted data faster, as all elements within the tree are sorted during in-ordered traversal. A Binary Search Tree also allows for fast searching and access to data, since the nodes are arranged based on their value, which results in half the remaining data to be removed from the search as it traverses each node.

**Recommendation:**

I recommend using a Binary Search Tree for this project. I do not believe I did the line-by-line Big O analysis right, so this recommendation is based on the advantages of each data structure and a Big-O Complexity Chart. We are looking to have the capability to print out an alphanumerically sorted course list based on the data. It would be easier to sort the data as it is inserted in a Binary Search Tree. As courses are added to the course list overtime you will easily be able to create new courses and insert them as needed, without unnecessary resizing. With courses constantly being added, changed, and removed from curriculums you would be unable to predict the size of data accurately which can make Hash Tables and Vectors challenging. A Binary Search Tree would also allow you to insert data into the middle of the structure, which would be beneficial for the type of data in this instance.