**Project One:**

**Pseudocode:**

**Vector-**

FUNCTION openRead(prequisite1, prequisite2)

         fstream file

         Open file

         IF file format contains errors

                     Exit

         END IF

         WHILE getting each line

         parse each line

                 FOR each line

                             IF line does not have at least two parameters

                                         Ignore line

                             ELSE

                                         Verify that the prequisites are found in the beginning of another line in the file

                             END IF

                     END FOR

         END WHILE

END FUNCTION

FUNCTION courseObject()

         Initialize vector

         fstream file

         Open file

         WHILE getting each line

                     Store the course object into the vector

         END WHILE

END FUNCTION

FUNCTION printCourseInformation(Vector<Course> courses, String courseNumber)

FOR all courses

                     IF the course is the same as courseNumber

                                 print out the course information

                     END IF

                                 FOR each prerequisite of the course

                                             print the prerequisite course information

                                 END FOR

         END FOR

END FUNCTION

FUNCTION displayMenu()

Print menu choices load data structure, print course list, print course, and exit

END FUNCTION

FUNCTION courseNumberComp(string1, string2)

Return string1 is less than string2

END FUNCTION

FUNCTION aplhaArrange(vector<Course> courseNumber, vector<Course> courses)

Sort course numbers using courseNumberComp()

FOR courses that contain courseNumber

Return sorted course vector

END FOR

END FUNCTION

FUNCTION main()

Pass the vector course numbers and courses through alphaArrange()

Print courses

END FUNCTION

**Hash Table-**

FUNCTION openRead(prequisite1, prequisite2)

         fstream file

         Open file

         IF file format contains errors

                     Exit

         END IF

         WHILE getting each line

         parse each line

                     FOR each line

                                 IF line does not have at least two parameters

                                             Ignore line

                                 ELSE IF the prerequisites are found in the beginning of another line in the file

                                                         Return course name, course number, and prerequisites to courseObject()

                                 END ELSE IF

                                 END IF

                     END FOR

         END WHILE

END FUNCTION

FUNCTION courseObject(courseName,courseNumber, prereq1, prereq2)

         Initialize hash table vector Vector<Course> courses

         Initialize new course node

         fstream file

         Open file

         WHILE node is not equal to null pointer

                     Store course name, course number, and prerequisites into the hash table

         END WHILE

END FUNCTION

FUNCTION printCourseInformation(Vector<Course> courses, String courseNumber)

FOR all courses in node list

                     IF the course is the same as courseNumber

                                 Display course information

                     END IF

                                 FOR each prerequisite of the course

                                             print the prerequisite course information

                                 END FOR

         END FOR

END FUNCTION

FUNCTION displayMenu()

Print menu choices load data structure, print course list, print course, and exit

END FUNCTION

FUNCTION aplhaArrange(vector<Course> courseNumber, vector<Course> courses)

Sort course numbers using vectors sorting method

FOR vectors courses that start with course number

Return sorted course vector

END FOR

END FUNCTION

FUNCTION main()

Pass the vector course numbers and courses through alphaArrange()

Print courses

END FUNCTION

**Binary Tree-**

Initialize root node

Initialize left node

Initialize right node

Initialize Bid

STRUCT Bid

         Declare courseName

         Declare courseNumber

         Declare prerequisite1

         Declare prerequisite2

END STRUCT

FUNCTION openRead(prequisite1, prequisite2)

         fstream file

         Open file

         IF file format contains errors

                     Exit

         END IF

         WHILE getting each line

         parse each line

                     FOR each line

                                 IF line does not have at least two parameters

                                             Ignore line

                                 ELSE IF the prerequisites are found in the beginning of another line in the file

                                                         Pass courseNumber, courseName, prerequisite1, prequiste2 to courseObject() function

                                 END ELSE IF

                                 END IF

                     END FOR

         END WHILE

END FUNCTION

FUNCTION courseObject(courseNumber, courseName, prerequisite1, prequiste2)

         fstream file

         Open file

         WHILE node is not equal to null pointer

                     Set courseNumber to node courseNumber

Set courseName to node courseName

Set first prerequisite to node prerequisite1

Set second prerequisite to node prerequisite2

         END WHILE

END FUNCTION

FUNCTION printCourseInformation(courseNumber)

         Set current node equal to root

WHILE current node is not equal to null pointer

                     IF the current courseNumber is the same as courseNumber

                                 Print course information

                     END IF

                                 FOR each prerequisite of the course

                                             Print the prerequisite course information

                                 END FOR

         END WHILE

END FUNCTION

FUNCTION displayMenu()

Print menu choices load data structure, print course list, print course, and exit

END FUNCTION

FUNCTION aplhaArrange(vector<Course> courseNumber, vector<Course> courses)

WHILE next node is not equal to null pointer

IF current course number is less than next course number

Return course number

ELSE

Switch nodes

END IF

END WHILE

END FUNCTION

FUNCTION main()

Pass the vector course numbers and courses through alphaArrange()

Print courses

END FUNCTION

**Evaluation:**

**Vector Runtime Analysis:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| fstream file | 1 | 1 | 1 |
| Open file | 1 | 1 | 1 |
| IF file format contains errors | 1 | 1 | 1 |
| Exit | 1 | 1 | 1 |
| WHILE getting each line | 1 | n | n |
| Parse each line | 1 | n | n |
| FOR each line | 1 | n | n |
| IF line does not have at least two parameters | 1 | n | n |
| Ignore line | 1 | 1 | 1 |
| ELSE Verify that the prequisites are found in the beginning of another line in the file | 1 | n | n |
| Initialize vector | 1 | 1 | 1 |
| Fstream file | 1 | 1 | 1 |
| Open file | 1 | 1 | 1 |
| While getting each line | 1 | n | n |
| Store the course object into the vector | 1 | n | n |
| **Total Cost** | | | 7n + 8 |
| **Runtime** | | | O(n) |

**Hash Table Runtime Analysis:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| fstream file | 1 | 1 | 1 |
| Open file | 1 | 1 | 1 |
| IF file format contains errors | 1 | 1 | 1 |
| Exit | 1 | 1 | 1 |
| WHILE getting each line | 1 | n | n |
| Parse each line | 1 | n | n |
| FOR each line | 1 | n | n |
| IF line does not have at least two parameters | 1 | n | n |
| Ignore line | 1 | 1 | 1 |
| ELSE IF the prerequisites are found in the beginning of another line in the file | 1 | n | n |
| Return course name, course number, and prerequisites to courseObject() | 1 | 1 | 1 |
| Initialize hash table vector Vector<Course> courses | 1 | 1 | 1 |
| Initialize new course node | 1 | 1 | 1 |
| Fstream file | 1 | 1 | 1 |
| Open file | 1 | 1 | 1 |
| WHILE node is not equal to null pointer | 1 | n | n |
| Store the course object into the vector | 1 | n | n |
| **Total Cost** | | | 7n + 10 |
| **Runtime** | | | O(1) |

**Binary Tree Runtime Analysis:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| fstream file | 1 | 1 | 1 |
| Open file | 1 | 1 | 1 |
| IF file format contains errors | 1 | 1 | 1 |
| Exit | 1 | 1 | 1 |
| WHILE getting each line | 1 | n | n |
| Parse each line | 1 | n | n |
| FOR each line | 1 | n | n |
| IF line does not have at least two parameters | 1 | n | n |
| Ignore line | 1 | 1 | 1 |
| ELSE IF the prerequisites are found in the beginning of another line in the file | 1 | n | n |
| Pass courseNumber, courseName, prerequisite1, prequiste2 to courseObject() function | 1 | 1 | 1 |
| Initialize vector | 1 | 1 | 1 |
| Fstream file | 1 | 1 | 1 |
| Open file | 1 | 1 | 1 |
| While getting each line | 1 | n | n |
| Set courseNumber to node courseNumber | 1 | 1 | 1 |
| Set courseName to node courseName | 1 | 1 | 1 |
| Set first prerequisite to node prerequisite1 | 1 | 1 | 1 |
| Set second prerequisite to node prerequisite2 | 1 | 1 | 1 |
| **Total Cost** | | | 6n + 13 |
| **Runtime** | | | O(log n) |

**Advantages and disadvantages:**

The data structure's vectors, binary trees, and hash tables all have their advantages and disadvantages. Vectors are dynamic in size meaning that, unlike arrays, they can change in size. This is great for programs that need to add or remove data from the vector throughout runtime but not great from a memory perspective. Other data structures such as lists have a predefined amount of storage allocation when declared. Hash tables are in constant time meaning that the runtime complexity for them is low. With hash tables, there might be issues with hash collisions meaning that two values have the same hash value. These can cause the program to run inefficiently and produce unexpected results. The main advantage of binary trees is simplicity since they are easy to understand. Like hash tables, they also have a low runtime complexity. Deleting nodes can be a complex process in a binary tree. This is a common operation so being able to do that efficiently is important from a development perspective.

**Recommendation:**

The data structure that I would recommend using for my project is the binary tree structure. Using the runtime analysis charts, you can see the differences in the data structure's big O notations. The three listed big O’s are O(n), O(1), and O(log n). Big O is used to show how fast an algorithm is. O(log n) and O(1) are considered excellent because they run the quickest. The reason I chose to use the binary tree in comparison to the hash table has to do with its total cost and advantages. The binary tree uses six repeating executions while the hash table uses seven. It is not a big difference but can reduce complexity in the code. Binary trees are also a simpler option to use than hash tables and are always sorted. Since the program has to do with the sorting of existing courses, I believe a binary tree would do a quick and efficient job of displaying the proper information.