1. **Resubmit pseudocode from previous pseudocode assignments and update as necessary**.

1A) Pseudocode to open file, read data from file, parse each line, and checks for formatting errors

Use fstream to open file ‘fileName.txt’

If the file can’t open

Return -1 to indicate file was unable to open

Exit Program

Create an empty vector for courses

Create an empty set for course numbers

While the file hasn’t reached the end of the file (EOF):

Read every line

Split each line of the course data via commas

If a line has less than two values

Return -1 as an Error for lack of sufficient parameters on the line

Otherwise, continue reading the files.

Set course number to course\_data[0]

Set course title to course\_data[1]

If course\_data length less than 2

Return as error

Else

Set course\_data[2:] for the prerequisites

1B) Pseudocode to create course objects so that the one course objects holds data from a single line from the input file (Copypasted pseudocodes from previous assignments)

//VECTOR PSEUDOCODE (Module 3-3)

Open the file

Define a struct ‘Course’ with the fields ‘course\_number’, ‘course\_’title’, and ‘course\_prerequisites’

Create courseList as an empty vector

Initilialize empty vector ‘courseVector’

While the file hasn’t reached the end of the file:

Split each line by commas

Create new Course class object newCourse

newCourse.course\_number = first token as course number

newCourse.course\_title = second token as course title

newCourse.course\_prerequisites = rest of the tokens as course prerequisites

Append newCourse to courseVector

Close the file

//HASH PSEUDOCODE (Module 4-3)

//Structure Defined in order to hold course information

STRUCT Course

String courseNumber

String courseTitle

ArrayList<String> coursePrerequisites

//Constructor to initialize Course Structure information

Course(string number, string title, ArrayList<string> prereq)

courseNumber = number

courseTitle = title

coursePrerequisites = prereq

HashTable <Course> courses = new HashTable()

//AFTER FILE IS READ FROM STEP ONE

Unsigned int key = Hash(newCourse.courseNumber)

IF courses[key] IS EMPTY

courses[key] = course

ELSE

Node currentNode = courses[key]

While current->next is NOT EQUAL TO nullptr

currentNode = current->next

current->next = newNode

//BINARYTREE PSEUDOCODE (Module 5-3)

//Structure defined to hold course information

STRUCT Course

String courseNumber

String courseTitle

ArrayList<String> coursePrerequisites

//Constructor to initialize Course Structure information

Course(String number, String title, ArrayList<String> prereq)

courseNumber = number

courseTitle = title

coursePrerequisites = prereq

//Structure defined node for BST

STRUCT Node

Course course

Node \*left

Node \*right

//Constructor to initialize node for BST

Node()

left = nullptr

right = nullptr

BinarySearchTable <Course> course = new BinarySearchTable()

//AFTER THE FILE HAS BEEN OPENED AND READ

FUNCTION BST insertNode(Node\* root, Course course)

IF root IS NULL

course.courseNumber = newCourse

course->left = NULL

course->right = NULL

RETURN courses

//Traverse root Node to the left or to the right for the following if-else statements

IF course->courseNumber IS LESS THAN root->course->courseNumber

root->left = insertNode(root->left, course)

ELSE

root->right = insertNode(root->right, course)

RETURN root

1C) Pseudocode to print out course information and prerequisites

//This pseudocode can work for vector, hash, and binaryTree functions altogether

ASSIGN userInput to key

For each key IN courses

IF key matches courses.courseNumber

PRINT courses.courseNumber and courses.CourseTitle

FOR each prereq of courses.courseNumber

PRINT every prereq course information

1. **Create pseudocode for a menu**.

PROGRAM STARTS

INITIALIZE int option to 0

WHILE option NOT EQUAL TO 9

IF option EQUALS 1

// 2A) LOAD file data into the data structure

OPEN file for reading

FOR each line in file

READ and split each line by comma

ADD course to the course list

CLOSE file

IF option EQUALS 2

// 2B) PRINT ordered list of all courses in order

//SORT courses in order

FOR LOOP int i less than courses size - 1

FOR LOOP int j less than courses size – i – 1

IF course index j less than course index j + 1

SET temp EQUALS course index j

SET course index j EQUALS course index j + 1

SET course index j + 1 EQUALS temp

//Iterate through ordered list and print

FOR course IN courses

PRINT course number and title

IF option EQUALS 3

// 2C) PRINT course title and prerequisites for any individual course

INPUT course title AS key

FOR each key IN courses

IF key EQUALS course title

PRINT course title

FOR each prerequisite of course title

PRINT prerequisites for the course

IF option EQUALS 9

// 2D) Break loop

EXIT WHILE LOOP

PROGRAM ENDS

1. **Design pseudocode that will print out the list of the courses in the Computer Science program in alphanumeric order.**

//FROM LOWEST TO HIGHEST

//Same pseudocode from Option 2 from (2.) but reversed

FOR LOOP int i less than courses size - 1

FOR LOOP int j less than courses size – i – 1

IF course index j **greater than** course index j + 1

SET temp EQUALS course index j + 1

SET course index j + 1 EQUALS course index j

SET course index j EQUALS temp

1. **Evaluate the run time and memory of data structures that could be used to address the requirements**.

**Vector Data Structure Time Complexity**

|  |  |  |  |
| --- | --- | --- | --- |
| **CODE** | **LINE COST** | **No. TIMES ITERATED** | **TOTAL COST** |
| GET userInput for course | 1 | 1 | 1 |
| FOR each course IN Courses | 1 | N | N |
| IF userInput matches course | 1 | N | N |
| PRINT course information | 1 | 1 | 1 |
| FOR every prerequisite | 1 | N | N |
| PRINT each requisite course information | 1 | 1 | 1 |
| PRINT course couldn’t be found if userInput couldn’t be found | 1 | 1 | 1 |
| Total Cost: | | | 3n + 1 |
| Runtime: | | | O(n) |

Total Cost: 3n + 1

Runtime: O(n)

**Hash Table Data Structure Time Complexity**

|  |  |  |  |
| --- | --- | --- | --- |
| **CODE** | **LINE COST** | **No. TIMES ITERATED** | **TOTAL COST** |
| ASSIGN unsigned int key to Hash course number | 1 | 1 | 1 |
| IF course key is EMPTY | 1 | N | N |
| course key EQUALS course | 1 | 1 | 1 |
| ELSE | 1 | 1 | 1 |
| currentNode equals course key | 1 | 1 | 1 |
| WHILE next node NOT EQUAL TO null pointer | 1 | 1 | 1 |
| Current Node equals the next node, current next node equals new node | 2 | N | 2N |
| Total Cost: | | | 3n+1 |
| Runtime: | | | O(n) |

**Binary Tree Data Structure Time Complexity**

|  |  |  |  |
| --- | --- | --- | --- |
| **CODE** | **LINE COST** | **No. TIMES ITERATED** | **TOTAL COST** |
| IF root IS NULL | 1 | N | N |
| SET courseNumber as new course  SET course left EQUAL NULL  SET course right EQUAL NULL  RETURN courses | 4 | N | 4N |
| IF courseNumber is less than the root course number | 1 | N | N |
| SET left root equal insertNODE (left root, course) | 1 | N | N |
| ELSE | 1 | 1 | 1 |
| SET right root equal insertNODE (right root, course) | 1 | N | N |
| Total Cost: | | | 8n + 1 |
| Runtime: | | | O(n) |

Every data structure’s runtime is O(n).

1. **Explain the advantages and disadvantages of each structure in your evaluation.**

* **Vector**
  + **Advantage**
    - It’s the easiest data structure to understand in comparison to the others.
    - When it’s runtime is O(1), adding items in the vector is sufficiently quick as well as retrieving the items.
    - Vectors can dynamically resize themselves whenever items are added or removed making it flexible for both growth and shrinkage.
  + **Disadvantage**
    - Vectors require more memory than they need to accommodate future growth. If not managed properly, it can lead to wasted space.
    - When the list is long, it’s speed can be relatively slow in comparison to the other two data structures.
* **Hash Table**
  + **Advantage**
    - It’s better than vectors in terms of adding items and searching for them.
    - It’s great for fast retrieving items especially at crucial times.
    - It can handle managing large amounts of data.
  + **Disadvantage**
    - Unlike vectors, implementing a Hash Table is rather complicated which requires time and patience to understand.
    - Collisions may occur when two hash keys have the same index.
    - Hash tables don’t sort their items in order.
* **Binary Tree**
  + **Advantage**
    - Binary Tree can be easier to understand versus Hash Tables.
    - It’s searches for items is faster than vectors.
    - Binary Trees store their data in an orderly fashion.
  + **Disadvantage**
    - It’s slower than Hash Tables.
    - Despite it being easier to understand than Hash Tables, it’s not less difficult than vectors.

1. **Make a recommendation for which data structure you plan to use in your code**

Having analyzed all three of the data structures, the one that I plan to use in my code is the Vector structure. I’m more confident in implementing that versus the other two. There may not be lots of classes to store so the runtime on that shouldn’t be a hassle. Though if vector wasn’t an option, my second recommendation would be Binary Search Tree for as I find it to be not too challenging to understand versus Hash Tables. Another thing about vectors is that they can resize data storage whether people want to grow more elements or shrink them. A resizable data structure can come in handy for situations like the data storage of classes and it’s prerequisite classes.