**Linked list reading file Pseudocode**

**START** program

**SET** string *csvPath* to course list file path

**CREATE** vector of courses named *courseList*

**TRY** to open path

**IF** file format is good

**SET** Parser file = Parser sending *csvPath* as the argument

**OPEN** csvPath

**WHILE** file is open

**ITERTERATE** through each line

**SPLIT** line into array by commas

**IF** current line size is equal to 2

**APPEND** course object to *courseList*

**ELSE IF** current line size is equal to 3 or 4

**ITERTERATE** through elements 3 and 4

**IF** element’s Number not in list

**EXIT** iteration

**ELSE**

**CONTINUE**

**ELSE**

**CONTINUE**

**CLOSE** file

**CATCH** error

**IF** file not open

**THROW** exception

**END** program

**Linked List Adding to Structure Pseudocode**

**START** program

**SET** string *csvPath* to course list file path

**SET** Parser file = Parser sending *csvPath* as the argument

**CREATE** vector of courses named *courseList*

**READ** in csvPath

**WHILE** not at end of file

**ITERATE** threw each line

**SPLIT** current line into elements in an array at commas

**CREATE** Course named *course*

**SET** course’s *courseNumber* equal to current lines first element

**SET** course’s *courseName* equal to current lines second element

**IF** *prerequisite1* does not exist

**SET** course’s *prerequisite1* equal to null

**ELSE**

**SET** course’s *prerequisite1* equal to current lines third element

**IF** *prerequisite2* does not exist

**SET** course’s *prerequisite2* equal to null

**ELSE**

**SET** course’s *prerequisite2* equal to current lines fourth element

**APPEND** to *courseList* sending *course*

**CLOSE** file

**END** program

**Linked List Print Pseudocode**

**START** program

**CREATE** string named *courseNumber* **SET** course number to search for

**CREATE** Course object named *course*

**CREATE** Node pointer named *currentNode* **SET** to head

**IF** *courseNumber* is equal to the *courseNumber* of the course head points too

**OUTPUT** the bid head points too

**WHILE** *currentNode* is not equal to a null pointer

**IF** currentNode’s *courseNumber* is equal to *courseNumber*

**OUTPUT** currentNode’s *course*

**ELSE**

**SET** *currentNode* equal to currentNode’s *next*

**OUTPUT** *course*

**END** program

**Hash Table Reading in File Pseudocode**

**START** program

**SET** string *csvPath* to course list file path

**CREATE** hash table of courses named *courseTable*

**TRY** to open path

**IF** file format is good

**SET** Parser file = Parser sending *csvPath* as the argument

**OPEN** csvPath

**WHILE** file is open

**ITERTERATE** through each line

**SPLIT** line into array by commas

**IF** current line size is equal to 2

**INVOKE** Insert() function sending currentcourse object to *courseTable*

**ELSE IF** current line size is equal to 3 or 4

**ITERTERATE** through elements 3 and 4

**IF** element’s Number not in *courseTable*

**EXIT** iteration

**ELSE**

**CONTINUE**

**ELSE**

**CONTINUE**

**CLOSE** file

**CATCH** error

**IF** file not open

**THROW** exception

**END** program

**Hash Table Adding to Structure Pseudocode**

**START** program

**SET** string *csvPath* to course list file path

**SET** Parser file = Parser sending *csvPath* as the argument

**CREATE** hashTable of courses named *courseTable*

**READ** in csvPath

**WHILE** not at end of file

**ITERATE** threw each line

**SPLIT** current line into elements in an array at commas

**CREATE** Course named *course*

**SET** course’s *courseNumber* equal to current lines first element

**SET** course’s *courseName* equal to current lines second element

**IF** *prerequisite1* does not exist

**SET** course’s *prerequisite1* equal to null

**ELSE**

**SET** course’s *prerequisite1* equal to current lines third element

**IF** *prerequisite2* does not exist

**SET** course’s *prerequisite2* equal to null

**ELSE**

**SET** course’s *prerequisite2* equal to current lines fourth element

**CREATE** an unsigned variable named *key*

**SET** key to the string value to of hash() function sending courses’ *courseNumber*

**CREATE** Node pointer named *preNode*

**SET** *preNode* to the pointer at node’s *key*

**IF** *preNode*’s key is equal to UINT\_MAX

**SET** *preNode*’s key to *key*

**SET** *preNode*’s course to *course*

**SET** *preNode*’s next to nullptr;

**ELSE**

**WHILE** *preNode*’s next is not equal to nullptr

**SET** *preNode* equal to *preNod*e’s next

**SET** *preNode*’s next equal to new node sending *course* and *key* as arguments

**CLOSE** file

**END** program

**Hash Table Print Pseudocode**

**START** program

**ITERATE** through *nodes* from beginning to end

**IF** the current *key* is not equal to UINT\_MAX

**OUTPUT** “KEY” + the current *key* + “: “ + the current *course*’s courseNumber

**OUTPUT** “ | “ + the current *course*’s title + “ | “ + the current *course’s* fund

**SET** Node pointer *node* equal to the current node’s next

**WHILE** *node* is not equal to nullptr

**OUTPUT** “KEY” + the current key + “: “ + the current *course*’s courseNumber

**OUTPUT**  “ | “ + the current *course*’s course name

**IF** course’s *prerequisite1* is not null

**OUTPUT** “ | “ + the current *course*’s prerequisite2

**IF** courses *prerequisite2* is not null

**OUTPUT** “ | “ + the current *course*’s prerequisite2

**SET** *node* equal to *node*’s next

**RETURN**

**END** program

**Binary Search Tree Read File Pseudocode**

**START** program

**SET** string *csvPath* to course list file path

**CREATE** BinarySearchTree() structure of courses named *courseBST*

**TRY** to open path

**IF** file format is good

**SET** Parser file = Parser sending *csvPath* as the argument

**OPEN** csvPath

**WHILE** file is open

**ITERTERATE** through each line

**SPLIT** line into array by commas

**IF** current line size is equal to 2

**INVOKE** Insert() function sending currentcourse object to *courseBST*

**ELSE IF** current line size is equal to 3 or 4

**ITERTERATE** through elements 3 and 4

**IF** element’s Number not in *courseBST*

**EXIT** iteration

**ELSE**

**CONTINUE**

**INVOKE** Insert() function sending currentcourse object to *courseBST*

**ELSE**

**CONTINUE**

**CLOSE** file

**CATCH** error

**IF** file not open

**THROW** exception

**END** program

**Binary Search Tree Adding to Structure Pseudocode**

**START** program

**SET** string *csvPath* to course list file path

**SET** Parser file = Parser sending *csvPath* as the argument

**CREATE** BinarySearchTree() structure of courses named *courseBST*

**READ** in csvPath

**WHILE** not at end of file

**ITERATE** threw each line

**SPLIT** current line into elements in an array at commas

**CREATE** Course named *course*

**SET** course’s *courseNumber* equal to current lines first element

**SET** course’s *courseName* equal to current lines second element

**IF** *prerequisite1* does not exist

**SET** course’s *prerequisite1* equal to null

**ELSE**

**SET** course’s *prerequisite1* equal to current lines third element

**IF** *prerequisite2* does not exist

**SET** course’s *prerequisite2* equal to null

**ELSE**

**SET** course’s *prerequisite2* equal to current lines fourth element

**CREATE** Bid named bid

**SET** *bid* using *courseNumber*, *courseName*, *prerequeisite1*, and *prerequeisite2*

**IF** course’s *courseNumber* < node’s *courseNumber*

**IF** node’s *left* is equal to a null pointer

**SET** node’s *left* equal to Node() function sending in *bid*

**ELSE**

**SET** pointer the node’s *left*

**ELSE**

**IF** node’s *right* is equal to null pointer

**SET** node’s *right* equal to Node() function sending in *bid*

**ELSE**

**SET** pointer the node’s *right*

**CLOSE** file

**END** program

**Binary Search Tree Print Pseudocode**

**START** program

**ITERATE** through *nodes* from beginning to end

**IF** current node is not equal to a null pointer

**INVOKE** inOrder() function sending in node’s *left* as an argument

**OUTPUT** current node’s course’s *courseNumber* + “ : “ + current node’s course’s *courseName*

**IF** *prerequiste1* is not null

**OUTPUT** “ | “ + current node’s course’s *prerequisite1*

**IF** *prerequiste2* is not null

**OUTPUT** “ | “ + current node’s course’s *prerequisite2*

**INVOKE** inOrder() function sending in node’s *right* as an argument

**RETURN**

**END** program

**Menu Pseudocode**

**START** program

**SET** integer *choice* = 0

**CREATE** a dataStructure object named *courseStruct*

**CREATE** vector *course* to store Course objects

**CREATE** time\_t *ticks*

**WHILE** *choice* not equal to 9

**OUTPUT** “1: Load Courses”

**OUTPUT** “2: Print Sorted Courses”

**OUTPUT** “3: Print Course”

**OUTPUT** “4: Exit”

**INPUT** *choice*

**IF** choice equals 1

**SET** *ticks* equal to the current system time

**INVOKE** loadCourses() function, sending csvPath,

**SET** *ticks* equal to current time - *ticks*

**OUTPUT** “time: “ + *ticks* + “ milliseconds”

**OUTPUT** “time: “ + *ticks* \* 1.0 / CLOCKS\_PER\_SEC + “ seconds”

**ELSE IF** choice equals 2

**SET** *ticks to* current system time

**INVOKE** printSorted()

**OUTPUT** “time: “ + *ticks* + “ clock ticks”

**OUTPUT** “time: “ + *ticks* \* 1.0 / CLOCKS\_PER\_SEC + “seconds”

**ELSE IF** *choice* equals 3

**SET** *ticks to* current system time

**INVOKE** Print()

**OUTPUT** “time: “ + *ticks* + “ clock ticks”

**OUTPUT** “time: “ + *ticks* \* 1.0 / CLOCKS\_PER\_SEC + “seconds”

**ELSE IF** *choice* equals 4

**EXIT** loop

**ELSE**

**CONTINUE** loop

**OUTPUT** “Good bye”

**END** program

**Linked List Sorted Print Pseudocode**

**START** program

**CREATE** a node pointer named *curNode* set to courseList head

**CREATE** vector of courses named *courses*

**LOOP** through courseList as long as current node is not null

**ADD** *curNode* to *courses*

**SET** *curNode* to curNode’s next

**CREATE** integer variable named *min*

**CREATE** integer variable named *size\_t* storing courses size (needed for loop in code)

**LOOP** through *courses*

**SET***min* to courses current element

**LOOP** through remaining elements

**IF** *courses* new element’s course number is less than *min*’s course number

**SET** *min* equal to courses new element

**IF** *min* is not equal to current element

**SWAP** current element with *min*

**LOOP** through *courses*

**OUTPUT** current elements *courseNumber* + “, “ + current elements *courseName*

**IF** current element’s *prerequisite1* is not null

**OUTPUT** “, “ + current elements *prerequisite1*

**IF** current element’s *prerequisite2* is not null

OUTPUT “, “ + current element’s *prerequisite2*

**END** program

**Hash Table Sorted Print Pseudocode**

**START** program

**CREATE** vector of courses named *courses*

**ITERATE** through *nodes* from beginning to end

**IF** the current *key* is not equal to UINT\_MAX

**ADD** current course to courses

**SET** Node pointer *node* equal to the current node’s next

**WHILE** *node* is not equal to nullptr

**ADD** current course to courses

**LOOP** through *courses*

**OUTPUT** current elements *courseNumber* + “, “ + current elements *courseName*

**IF** current element’s *prerequisite1* is not null

**OUTPUT** “, “ + current elements *prerequisite1*

**IF** current element’s *prerequisite2* is not null

**OUTPUT** “, “ + current element’s *prerequisite2*

**RETURN**

**END** program

**Binary Search Tree Sorted Print Pseudocode**

**START** program

**ITERATE** through *nodes* from beginning to end

**IF** current node is not equal to a null pointer

**INVOKE** inOrder() function sending in node’s *left* as an argument

**OUTPUT** current node’s course’s *courseNumber* + “ : “ + current node’s course’s *courseName*

**IF** *prerequiste1* is not null

**OUTPUT** “ | “ + current node’s course’s *prerequisite1*

**IF** *prerequiste2* is not null

**OUTPUT** “ | “ + current node’s course’s *prerequisite2*

**INVOKE** inOrder() function sending in node’s *right* as an argument

**RETURN**

**END** program

**LINKED LIST PRINT**

|  |  |  |  |
| --- | --- | --- | --- |
| Code | Line Cost | # Times executed | Total Cost |
| **for all courses** | 1 | 1 | 1 |
| **if the course is the same as courseNumber** | 1 | n | 1n |
| **print out the course information** | 1 | n | 1n |
| **for each prerequisite of the course** | 1 | 2n | 3n |
| **print the prerequisite course information** | 1 | n | 1n |
|  |  | Total Cost | 6n + 1 |
|  |  | Runtime | O(n) |

**HASH TABLE PRINT**

|  |  |  |  |
| --- | --- | --- | --- |
| Code | Line Cost | # Times executed | Total Cost |
| **for all courses** | 1 | n | 1n |
| **if key is not equal to UNIT MAX** | 1 | n | 1n |
| **print out the course information** | 1 | 1 | 1 |
| **While node is not equal to nullptr** | 1 | 2n | 3n |
| **print the prerequisite course information** | 1 | n | 1n |
|  |  | Total Cost | 6n + 1 |
|  |  | Runtime | O(n) |

**BINARY SEARCHCH PRINT**

|  |  |  |  |
| --- | --- | --- | --- |
| Code | Line Cost | # Times executed | Total Cost |
| **for all courses** | 1 | 1 | 1 |
| **if node is not equal to nullptr** | 1 | n | 1n |
| **print out the course information** | 1 | n | 1n |
| **for each prerequisite of the course** | 1 | 2n | 3n |
| **print the prerequisite course information** | 1 | n | 1n |
|  |  | Total Cost | 6n + 1 |
|  |  | Runtime | O(n) |

**Analysis Evaluation**

There are many different ways to code a program that has all the given requirements. Each kind of structure has its own advantages and disadvantages. Depending on what is being looked for by the user, different structures may be used. Some may make results more accurate while others try to hide the inaccuracy. Some structures may allow for faster runtime. Others may allow better memory usage.

A linked list uses dynamic memory and connects data in a fashion that is easily envisioned. This allows for sorting to happen easily and accurately (depending on the sort function). The dynamic structure allows size to be determined when the program is first run. This also allows for ease of adding and deleting data in the structure. Though accurate did to the possibility of being large, it could use a lot of memory for storage. It also may have a slower run speed do the being read in the big O notation O(n). That being said n would be the number of lines being read so if there are 5 lines the runtime is O(5).

Hash tables are a good suit for many cases. They are considered more space efficient when it comes to memory usage. Due to its structure, the speed to find any piece of data with it is considered to be very good. The big O notation that a hash table reads as O(1). This means that even if there are 5 lines to read it will still run as O(1). Even though it is fast it doesn’t keep any consistent order, making the structure complex.

The last structure to compare is a binary search tree. This structure is very well organized. This structure’s speed is labeled as O(log n). That would be if there is 5 lines to read then it would be O(log 5). The organization of this structure helps to distinguish what piece of data goes where in the structure depending on a numerical value. It is not considered very efficient because of the pointers that are required to hold the whole thing together.

For the program that reads in the courses, the binary search tree will be the best option. The program doesn’t need to add or delete any data after loading, therefore linked lists are not suited. Linked lists will also run slower. Hash tables are not very organized, so binary search trees would be best for this program. The binary search tree will be able to organize the data by course number. Because the data will be organized printing will be faster and easier.

Vahid, F. (2019). CS300: Data Structure and Algorithms. ZyBooks. Retrieved July 2, 2023 from <https://learn.zybooks.com/zybook/CS-300-X6110-OL-TRAD-UG.23EW6>