***Pseudocode for Menu***

**BEGIN DEFINITION**

CREATE a method called menu with no parameters:

PRINT “1. Load Data Structure”.

PRINT “2. Print Course List”.

PRINT “3. Print Course”.

PRINT “4. Exit”.

CREATE variable called choice and SET to GET user input.

CREATE switch and pass in choice:

SET case 1:

CALL loadCourses with appropriate data structure.

SET case 2:

CALL print course method selected.

SET case 3:

SET variable courseNumber and SET to GET user input.

CALL print course information and pass in the courseNumber.

SET case 4:

EXIT program.’

END SWITCH

**END DEFINITION**

***Pseudocode for alphanumeric print***

*Vector*

**BEGIN DEFINITION**

CREATE method called printSortedVectorList and assign no parameters:

CREATE a variable called sortedVector and assign the sort method and assign it vector.

FOR the course in sortedVector:

DISPLAY the course

END LOOP

**END DEFINITION**

*Hash Table*

**BEGIN DEFINITION**

CREATE method called printSortedListHash and assign no parameters:

CREATE variable courseList and set to hashTable.values with no parameters.

CREATE variable sortedList and set to sortMethod and pass in courseList.

FOR course in sortedList:

DISPLAY Course

END LOOP

**END DEFINITION**

*Binary Tree*

**BEGIN DEFINITION**

CREATE method called printCourseListTree and assign no parameters:

CREATE variable sortedList and set to tree.inOrderTraversal with no parameters.

FOR course in sortedList:

DISPLAY COURSE

END LOOP

**END DEFINITION**

***Complete Pseudocode for Vector***

**Begin Definition**

CREATE a string variable called number.

CREATE a string variable called description.

CREATE a Vector<string> called prerequisites.

CREATE a Constructor that takes a vector of strings and processes them into a course class instance:

SET variable to fields[0];

SET description to fields[1];

FOR a variable called index SET to 2, index to the number of fields, update by 1:

APPEND fields[index] to prerequisites;

END FOR

END CONSTRUCTOR

**END DEFINITION**

**BEGIN DEFINITION**

CREATE method named ReadCourseFile:

CREATE a vector of strings called fields to contain the elements of a course description.

OPEN the provided file name, if an error occurs, return the perror value to the caller.

CLEAR courses vector.

WHILE there are more lines in the file to read:

READ a line of text, storing the text in a string variable called courseData.

If an error occurs while reading the line, close the file and return the perror string.

CREATE a temp string to hold the field name.

FOR EACH character in courseData:

IF the character is a comma:

APPEND the temp string to the fields vector.

CLEAR the temp string.

ELSE

ADD the character to the temp string.

END FOR

CALL the Course constructor to process the fields into a course instance.

Course course = Course(fields);

APPEND the course instance to the courses list.

END WHILE

CLOSE the file.

RETURN “Success”;

**END DEFINITION** ReadCourseFile

**BEGIN DEFINITION**

CREATE method called printSortedVectorList and assign no parameters:

CREATE a variable called sortedVector and assign the sort method and assign it vector.

FOR the course in sortedVector:

DISPLAY the course

END LOOP

**END DEFINITION**

**BEGIN DEFINITION**

CREATE a method called menu with no parameters:

PRINT “1. Load Data Structure”.

PRINT “2. Print Course List”.

PRINT “3. Print Course”.

PRINT “4. Exit”.

CREATE variable called choice and SET to GET user input.

CREATE switch and pass in choice:

SET case 1:

CALL loadCourses with appropriate data structure.

SET case 2:

CALL print course method selected.

SET case 3:

SET variable courseNumber and SET to GET user input.

CALL print course information and pass in the courseNumber.

SET case 4:

EXIT program.’

END SWITCH

**END DEFINITION**

**BEGIN DEFINITION**

CREATE the main method:

CALL menu.

RETURN 0.

**END DEFINITION**

***Complete Pseudocode for Hash Table***

**BEGIN DEFINITION**

string courseNum;

string courseDescription;

Vector<string> prerequisites;

START constructor inputs Vector<string>fields:

couseNum = fields[0];

courseDescription = fields[1];

FOR index = 2, for number of fields in fields, update index by one:

ADD fields[index] to prerequisites;

END FOR

END constructor

**END DEFINITION**

CREATE a vector called courses.

**BEGIN DEFINITION**

int element;

string number;

**END DEFINITION**

**BEGIN DEFINITION**

SET constant HASHMAP\_SIZE to 13.

CREATE an array called buckets of size HASHMAP\_SIZE for each element containing a vector of type Bucket.

CREATE a method that allows a course and its information to be added to the hashmap.

BEGIN ADD method INPUTS: string number, int courses index:

GET has value by calling the hash function and passing it the course number.

CREATE a new Bucket instance by calling its constructor and providing the course number and courses index.

ADD the new Bucket instance to the buckets away[hash value].

END ADD DEFINITION

CREATE a get method that takes a course number as input and returns the index in the courses list.

BEGIN definition GET INPUTS: string course courseNum:

GET hash value by calling the hash function.

FOR EACH bucket in buckets[hash value]:

IF bucket. courseNum == courseNum:

RETURN bucket.element;

END IF

RETURN -1;

END GET DEFINITION

CREATE a private hash method called HASH that takes a string value as input and returns an integer within the range of 0 and HASHMAP\_SIZE – 1.

BEGIN definition HASH INPUTS: string number:

SET hashValue = 0;

FOR EACH character in number:

hashValue += INT number;

hashValue %= HASHMAP\_SIZE;

RETURN hashValue;

END HASH DEFINITION

**END DEFINITION**

CREATE a hashmap instance called hashmap.

**BEGIN DEFINITION** ReadCoursesFile

CREATE a vector called fields that contains the course description.

OPEN the file, if an error occurs return a perror value.

CLEAR courses vector.

WHILE there are more lines in the file:

READ a line of text and store it in the string variable courseData.

IF an error occurs while reading the file, close the file and return a perror string.

CREATE a temp string to hold the field name.

FOR EACH character in the list:

IF the character is a comma:

ADD the temp string to the fields vector.

CLEAR the temp string.

ELSE

ADD the character to the temp string.

END FOR EACH

IF the number of strings in fields is less than 2, close file and return “Course number” + fields[0] + “’s information is invalid.”

CALL the course constructor to process fields into course instance.

Course course = course(fields)

GET the courses element number for the new course from the courses list.

ADD the course instance to the courses list.

ADD the course number and course element to the hash map by calling the Add method and passing the course instance number and courses element number.

END WHILE

CLOSE file

FOR EACH course in courses:

FOR EACH prerequisite in course.prerequisites:

IF hashmap.Get(prerequistes) < 0:

RETURN “Prerequiste course ” + prerequisite + “is not an offered course.”

END FOR EACH

END FOR EACH

RETURN “Success”;

**END Definition** ReadCoursesFile

**BEGIN DEFINITION**

CREATE method called printSortedListHash and assign no parameters:

CREATE variable courseList and set to hashTable.values with no parameters.

CREATE variable sortedList and set to sortMethod and pass in courseList.

FOR course in sortedList:

DISPLAY Course

END LOOP

**END DEFINITION**

**BEGIN DEFINITION**

CREATE a method called menu with no parameters:

PRINT “1. Load Data Structure”.

PRINT “2. Print Course List”.

PRINT “3. Print Course”.

PRINT “4. Exit”.

CREATE variable called choice and SET to GET user input.

CREATE switch and pass in choice:

SET case 1:

CALL loadCourses with appropriate data structure.

SET case 2:

CALL print course method selected.

SET case 3:

SET variable courseNumber and SET to GET user input.

CALL print course information and pass in the courseNumber.

SET case 4:

EXIT program.’

END SWITCH

**END DEFINITION**

**BEGIN DEFINITION**

CREATE the main method:

CALL menu.

RETURN 0.

**END DEFINITION**

***Complete Pseudocode for Binary Tree***

**BEGIN DEFINITION**

string courseNum;

string courseDescription;

Vector<string> prerequisites;

START constructor inputs Vector<string>fields:

couseNum = fields[0];

courseDescription = fields[1];

FOR index = 2, for number of fields in fields, update index by one:

ADD fields[index] to prerequisites;

END FOR

END constructor

**END DEFINITION**

CREATE a structure and call it Directions:

COURSE course;

CREATE a node called left.

CREATE a node called right.

SET Directions:

SET left equal to a null pointer.

SET right to a null pointer.

CREATE a class called BinaryTree:

CREATE a private node called root.

SET private methods to orderSorted, orderPost, and orderPre and assign a new node called node to them.

SET a new node for removeNode to node and the string of courseNum.

SET a public method called BinaryTreeSearch and SET the variables to a node called searchNode and a string called course.

SET a public method called InsertCourse and SET the variables to Course courseNum.

SET a public method called RemoveCourse and SET the variable to a string called courseNum.

SET public methods for orderSorted, orderPost, and orderPre.

**BEGIN DEFINITION** default constructor of BinaryTree:

SET root to a null pointer.

**END DEFINITION**

**BEGIN DEFINTION** orderSorted:

SET as orderSorted and pass in root.

**END DEFINITION**

**BEGIN DEFINITION** orderPost method:

SET orderPost and pass in root.

**END DEFINITION**

**BEGIN DEFINITION** orderPre method:

SET orderPre and pass in root.

**END DEFINITION**

CREATE a vector called courses

**BEGIN DEFINITION**

IF root is equal to null:

SET root to a new node.

SET root->courseNum to courseNum.

ELSE

Add a new node and assign it to both root and courseNum.

END IF

**END DEFINITION**

**BEGIN DEFINITION**

SET root and use removeNode at root on the courseNum.

**END DEFINITION**

***BEGIN DEFINITION***

*CREATE a variable called currentCourse and set it to the root.*

*WHILE the currentCourse is not equal to a null pointer:*

*IF the currentCourse’s courseNum is equal to the courseNum:*

*RETURN the currentCourse’s course.*

*END IF*

*IF the courseNum is less than the currentCourse’s courseNum:*

*SET currentCourse to the currentCourse’s left.*

*ELSE:*

*SET currentCourse to the currentCourse’s right.*

*END IF*

*END LOOP*

*CALL Course course.*

*RETURN course.*

***END DEFINITION***

CREATE a binary search tree instance called binaryTreeSearch.

**BEGIN DEFINITION**

IF the course.courseNum is less than the node’s course.courseNum:

IF the node to the left is equal to null:

Node to the left is equal to a new node.

Node to the left for course is equal to course.

ELSE:

Add a node for the node’s left using the course.

ELSE:

IF the node to the right is equal to null:

The right node needs to be set to a new node.

The node to the right’s course is equal to the course.

ELSE:

Add a new node to the right with the course info.

END IF

**END DEFINITION**

**BEGIN DEFINITION**

IF the course is not equal to null:

orderSorted is assigned left.

Then SET orderSorted right.

END IF

**END DEFINITION**

**BEGIN DEFINTION**

IF the courseNum is not equal to null:

SET orderPost to course left.

SET orderPost to course right.

END IF

**END DEFINITION**

**BEGIN DEFINTION**

IF the course is not equal to null:

SET orderPre to course left.

SET orderPre to course right.

END IF

**END DEFINITION**

**BEGIN DEFINTION**

IF course is equal to null:

RETURN course.

END IF

IF courseNum is less than the course’s course.courseNum:

The course to the left is removed.

ELSE IF the courseNum is greater than the course.courseNum:

The course to the right is removed.

ELSE:

If the course to the left is equal to null:

CREATE a Course called temp and store the course’s right value to it.

DELETE the course.

RETURN temp.

ELSE IF the node to the right is equal to null:

CREATE a Course called temp and store the course’s left value to it.

DELETE the course.

RETURN temp.

CREATE a temp variable and store course’s right value in it.

SET course’s courseNum to temp’s courseNum.

REMOVE course’s right value and replace with the temp’s value.

END IF

RETURN course.

**END DEFINITION**

**BEGIN DEFINITION** ReadCoursesFile

CREATE a vector called fields that contains the course description.

OPEN the file, if an error occurs return a perror value.

CLEAR courses vector.

WHILE there are more lines in the file:

READ a line of text and store it in the string variable courseData.

IF an error occurs while reading the file, close the file and return a perror string.

CREATE a temp string to hold the field name.

FOR EACH character in the list:

IF the character is a comma:

ADD the temp string to the fields vector.

CLEAR the temp string.

ELSE

ADD the character to the temp string.

END FOR EACH

IF the number of strings in fields is less than 2, close file and return “Course number” + fields[0] + “’s information is invalid.”

CALL the course constructor to process fields into course instance.

Course course = course(fields)

GET the courses element number for the new course from the courses list.

ADD the course instance to the courses list.

ADD the course number and course element to the hash map by calling the Add method and passing the course instance number and courses element number.

END WHILE

CLOSE file

FOR EACH course in courses:

FOR EACH prerequisite in course.prerequisites:

IF binaryTreeSearch.Get(prerequistes) < 0:

RETURN “Prerequiste course ” + prerequisite + “is not an offered course.”

END FOR EACH

END FOR EACH

RETURN “Success”;

**END Definition** ReadCoursesFile

**BEGIN DEFINITION**

CREATE method called printCourseListTree and assign no parameters:

CREATE variable sortedList and set to tree.inOrderTraversal with no parameters.

FOR course in sortedList:

DISPLAY COURSE

END LOOP

**END DEFINITION**

**BEGIN DEFINITION**

CREATE a method called menu with no parameters:

PRINT “1. Load Data Structure”.

PRINT “2. Print Course List”.

PRINT “3. Print Course”.

PRINT “4. Exit”.

CREATE variable called choice and SET to GET user input.

CREATE switch and pass in choice:

SET case 1:

CALL loadCourses with appropriate data structure.

SET case 2:

CALL print course method selected.

SET case 3:

SET variable courseNumber and SET to GET user input.

CALL print course information and pass in the courseNumber.

SET case 4:

EXIT program.’

END SWITCH

**END DEFINITION**

**BEGIN DEFINITION**

CREATE the main method:

CALL menu.

RETURN 0.

**END DEFINITION**

***Runtime Analysis/Advantages and disadvantages***

*Vector Structure*

|  |  |  |  |
| --- | --- | --- | --- |
| Operation | Cost Per Line | Number of times Executed | Big O Value |
| Loading Courses | 1 | O(n) | O(n) |
| Printing Sorted List | O(n log n) | O(n) | O(n log n) |
| Memory | 1 | O(n) | O(n) |

The advantages are: sequential access of data, work when there is more complex information, machine learning, and they are fast for certain types of searches. The disadvantages are: not great when searching becomes more complex, are a bit limited for certain data, and depend on vector representations.

*Hash Table*

|  |  |  |  |
| --- | --- | --- | --- |
| Operation | Cost Per Line | Number of times Executed | Big O Value |
| Loading Courses | 1 | O(n) | O(n) |
| Printing Sorted List | O(n log n) | O(n) | O(n log n) |
| Memory | 1 | O(n) | O(n) |

The advantages are: quick retrieval of data, insertion and deletion within the table is easier with only one array index to update, can store any type of data, and has resolution methods for when one or more data have the same key. The disadvantages are: null values are not allowed, too many collisions reduce efficiency, and harder to implement.

*Binary Tree*

|  |  |  |  |
| --- | --- | --- | --- |
| Operation | Cost Per Line | Number of times Executed | Big O Value |
| Loading Courses | 1 | O(n) | O(n) |
| Printing Sorted List | 1 | O(n) | O(n) |
| Memory | 1 | O(n) | O(n) |

The advantages are: efficient searching algorithm, efficient sorting, can be stored in a database, and implement decisions. The disadvantages are limited to 2 child nodes, can be unbalanced, can have a slower worst-case scenario when more complex.

***Recommendation***

Based upon these different structures’ runtime analysis and their advantages and disadvantages, while taking into account the task we have been asked to complete, I would recommend using the binary tree search structure because it is the most efficient in this case.

**Citations**

Mudadla, S. (2023, October 30). *What is vector database?advantages*. Medium. https://medium.com/@sujathamudadla1213/what-is-vector-database-advantages-e637fa7b80ef

aayushi2402, shreyasnaphad, & ikhilbhoi9739. (2023, March 28). *Applications, advantages and disadvantages of hash data structure*. GeeksforGeeks. https://www.geeksforgeeks.org/applications-advantages-and-disadvantages-of-hash-data-structure/

shreyasnaphad, shruti456rawal, reshmapatil2772, & raiankitsr. (2023, March 29). *Applications, advantages and disadvantages of Binary Tree*. GeeksforGeeks. https://www.geeksforgeeks.org/applications-advantages-and-disadvantages-of-binary-tree/