**Project One**

Leslie O’Boyle

8/10/2025

***MILESTONE PSEUDOCODE***

**Vector**

**File Input (into data structure)**

START

DEFINE vector<Course> StoreFile (vector<Course> courses)

OPEN file with ifstream

IF file is not open

PRINT Error opening file

RETURN

DECLARE new course object with no arguments (empty course object)

WHILE parsing the file line by line to the end of the file

DECLARE new stringstream to read file line

DECLARE string variable ‘token’

DECLARE string vector ‘tokens’

WHILE split the line using stringstream and ‘, ‘ as the separator into tokens

PUSH\_BACK token into tokens vector

IF tokens vector length is less than 2

PRINT error with file line parameters

RETURN

SET course object coursed attribute to first token

SET course object course name attribute to second token

IF size of vector is greater than 2

CREATE new list ‘prerequisites’

FOR every token from the third token to the end of the tokens vector

PUSH\_BACK token into prerequisites

CLOSE file

DECLARE Boolean variable find and set to false

FOR every course in the course vector

IF prerequisite list is not empty

FOR every prerequisite in the prerequisite list of the course

SET find to false

FOR every course in the course vector

IF the prerequisiteID matches courseID

SET find to true

IF find is FALSE

PRINT error: prerequisiteID not in list of courseIDs

END

**Course Object (uses file input method to store in data structure)**

START

DEFINE struct Course

DECLARE public/private attributes/methods (course ID, name, list of prerequisites)

DEFINE empty and parameterized constructors

DECLARE new vector of course objects

CALL file input method to input file into the vector with the new vector as an argument

END

**Print Course Information**

START

DEFINE void PrintCourses(vector<Course> courses, string courseID)

FOR every course in the courses vector

PRINT course ID

PRINT course name

FOR each prerequisite in the list of prerequisites

PRINT prerequisite

END

**Hash Table**

**File Input (into data structure)**

START

DEFINE vector<Course> StoreFile (vector<Course> courses)

OPEN file with ifstream

IF file is not open

PRINT Error opening file

RETURN

DECLARE new course object with no arguments (empty course object)

WHILE parsing the file line by line to the end of the file

DECLARE new stringstream to read file line

DECLARE string variable ‘token’

DECLARE string vector ‘tokens’

WHILE split the line using stringstream and ‘, ‘ as the separator into tokens

PUSH\_BACK token into tokens vector

IF tokens vector length is less than 2

PRINT error with file line parameters

RETURN

SET course object course attribute to first token

SET course object course name attribute to second token

IF size of vector is greater than 2

CREATE new list ‘prerequisites’

FOR every token from the third token to the end of the tokens vector

PUSH\_BACK token into prerequisites

SET unsigned int key to multiplicative string hash for object course number

IF node at key in course vector is a nullptr

SET node at key in vector to new node with bid information

ELSE IF node at key is UINT\_MAX

SET node attributes to parameter course attributes

ELSE

WHILE node is not a nullptr

SET node to next node

SET node next to new node with parameter course attributes

CLOSE file

DECLARE Boolean variable find and set to false

FOR every course in the course vector

WHILE course is not a nullptr

IF prerequisite list is not empty

FOR every prerequisite in the prerequisite list of the course

SET find to false

FOR every course in the course vector

IF the prerequisiteID matches courseID

SET find to true

IF find is FALSE

PRINT error: prerequisiteID not in course list

SET course to course’s next

END

**Course Object (uses file input method to store in data structure)**

START

DEFINE struct Course

DECLARE public/private attributes/methods ( string course ID, string name, list of string prerequisites, Course pointer next)

DEFINE empty and parameterized constructors

DECLARE new vector of course objects

CALL file input method to input file into the vector with the new vector as an argument

END

**Print Course Information**

START

DEFINE void PrintCourses(vector<Course> courses, string courseID)

FOR every course in the courses vector

WHILE course is not a nullptr

PRINT course ID

PRINT course name

FOR each prerequisite in the list of prerequisites

PRINT prerequisite

SET course to course’s next

END

**Binary Search Tree**

**File Input (into data structure)**

DEFINE Tree<Course> courses StoreFile (Tree<Course> courses)

OPEN file with ifstream

IF file is not open

PRINT Error opening file

RETURN

DECLARE new BST

DECLARE new course object with no arguments (empty course object)

WHILE parsing the file line by line to the end of the file

DECLARE new stringstream to read file line

DECLARE string variable ‘token’

DECLARE string vector ‘tokens’

WHILE split the line using stringstream and ‘, ‘ as the separator into tokens

PUSH\_BACK token into tokens vector

IF tokens vector length is less than 2

PRINT error with file line parameters

RETURN

SET course object course attribute to first token

SET course object course name attribute to second token

IF size of vector is greater than 2

CREATE new list ‘prerequisites’

FOR every token from the third token to the end of the tokens vector

PUSH\_BACK token into prerequisites

CALL addNode for adding bid onto root node

CALL parsePrereqs with root node as argument

DECLARE BinarySearchTree BST<course>

DECLARE private and public attributes and methods (head node)

void BST::parsePrereqs(Node\* node)

IF node is nullptr or node prereqs list is empty

RETURN

CALL parsePrereqs recursively for left child

CALL checkPrereqs for node

CALL parsePrereqs recursively for right child

void BST::checkPrereqs(Node\* node)

FOR each prerequisite in the node’s list of prerequisites

DECLARE Boolean find and set to false

DECLARE currNode and set to root node

WHILE find is false

IF node is nullptr

RETURN

ELSE IF node’s courseID is equal to the current prerequisiteID

SET find to true

ELSE

If prerequisiteID is less than courseID

SET currNode to left child

ELSE

SET currNode to right child

IF find is false

PRINT error: prerequisiteID not in course list

void BST::addNode(Node\*node, Course course)

IF BST courses’ root node is nullptr

SET root node to course

ELSE IF node’s course number is greater than the new course number

IF node’s left child is nullptr

SET left child to new course

ELSE

CALL addNode recursively for left child

ELSE

IF node’s right child is nullptr

SET right child to new course

ELSE

CALL addNode recursively for right child

END

**Course Object (uses file input method to store in data structure)**

START

DEFINE struct Course

DECLARE public/private attributes (string course ID, string name, list of string prerequisites)

DEFINE empty and parameterized constructors

DEFINE struct Node

DECLARE public/private attributes (Course\* course, Node\* left, Node\* right)

DEFINE empty and parameterized constructors

CALL file input method to input file into the vector with the new vector as an argument

END

**Print Course Information**

START

void printCourse (Node\* currNode, string courseID)

DECALRE boolean find to false

WHILE find is false

IF currNode is not nullptr

IF currNode courseID is equal to searched course ID

PRINT current course attribute information

SET find to true

ELSE IF

CALL printCourse recursively for left node

CALL printCourse recursively for right node

IF fine is false

PRINT course not found

END

***MENU PSEUDOCODE***

START

DEFINE void Menu()

DECLARE and SET integer choice to 0

WHILE the choice is not 9

PRINT the menu for the user

INPUT the user choice into choice

SWITCH for choice

CASE choice is 1

CALL StoreFile function to store files into a binary search tree

CASE choice is 2

CALL inOrder function to print all the courses in order

CASE choice is 3

CALL printCourse with head node of binary search tree and

searched course as argument

PRINT Goodbye

***PRINT ORDERED LIST PSEUDOCODE***

DEFINE void BinarySearchTree::inOrder(Node\* node)

IF current node is not nullptr

CALL inOrder recursively for left node

PRINT current node information

CALL inOrder recursively for right node

***RUNTIME AND MEMORY***

|  |  |  |
| --- | --- | --- |
|  | **Line Cost**  i= num prerequisites  n = num of file lines | **Worst Case Big O Value** |
| **Vector** | 2i + n + 14 | O(n) |
| **Hash Table** | 2i + n + 20 | O(n) |
| **Binary Search Tree** | 2i + 2n + 21 | O(n) |

1. Based on the advisor’s requirements, analyze each of the vector, hash table, and tree data structures. **Explain the advantages and disadvantages of each structure in your evaluation.**
2. Now that you have analyzed all three data structures, **make a recommendation for which data structure you plan to use in your code**. Provide justification for your recommendation based on the Big O analysis results and your analysis of the three data structures.

***ADVANTAGES AND DISADVANTAGES***

A line cost and worst case Big O value is shown for each data structure in the table above. The line cost for every data structure contains loops for both the number of file lines and number of prerequisites. Because of this, two variables are used to depict the difference variables that could impact that line cost. The calculation for each data structure is shown in the table. The worst case Big O value for all of the data structures is O(n). For the vector, each item is added one at a time, summing to n. Therefore, the time complexity at worse case would be O(n). The same can be said for both the hash table and the binary search tree data structures. The total line cost is highest for a binary search tree, followed by the hash table, then lastly the vector data structure.

Vectors, unlike arrays, dynamically resize themselves as elements are added or removed. This flexibility enables efficient memory management and element manipulation. Vectors can support various data types, and have built-in mechanisms to prevent errors. Sorting arrays by element value can be timely and end in a high line cost, making printing in order less efficient than other data structures (Myscale, 2024).

Hash data structures are used to store and retrieve records quickly. For example, a database might use a hash to index records by a unique identifier such as a social security number or customer ID. Use space more efficiently than BST, which require more memory to store pointers. Hashes are inefficient when there are many collisions. Also, hash tables have a limited capacity and will eventually fill up. They also do not maintain the order of elements, which make it difficult to retrieve elements in a specific order (GeeksforGeeks, 2025a).

Binary search trees are efficient in searching, and have an ordered structure that allows for accessing the elements in a specific order. They allows for dynamic insertions and deletions on the data stored in the tree. Although, in the worst case, BSTs can have a linear time complexity for searching and insertion. They can become inefficient for very large datasets, and have limited functionality (GeeksforGeeks, 2025b).

***Recommendation***

For this situation, I would recommend a binary search tree data structure. This data structure allows access to the data in an ordered fashion for printing, which is necessary for this application. It has a worse case Big O value of O(n), which is no better or worse than either of the data structures. BSTs allow for easy insertions and deletions, allowing for courses to be added and removed whenever the school needs.

**References**

GeeksforGeeks. (2025, July 23a). *Applications, Advantages and Disadvantages of Binary*

*Search Tree.* [*https://www.geeksforgeeks.org/dsa/applications-advantages-and-*](https://www.geeksforgeeks.org/dsa/applications-advantages-and-)

*disadvantages-of-binary-search-tree/*

GeeksforGeeks. (2025, July 23b). *Applications, Advantages and Disadvantages of Hash*

*Data Structure.* <https://www.geeksforgeeks.org/dsa/applications-advantages-and->disadvantages-of-hash-data-structure/

Myscale. (2024, April 02). *5 Key Advantages of Using Vectors in C++ Programming.*

<https://myscale.com/blog/advantages-vectors-cpp-programming/>