Project One

Pseudocodes

Vector Data Structure

START

INCLUDE FSTREAM library to open and read file

ADD STRING library

ADD UTILITY library

CREATE course STRUCTURE

    DECLARE courseNumber, courseName, and coursePrereq vectors with STRING data type

    DEFINE all three above vectors to PUSH_BACK user input

    DEFINE all three above vectors to separate data in each vector using “,”

END STRUCTURE

CREATE function to load, read, and parse course file

    DECLARE variable line

    CREATE object of ifstream class to OPEN course file

    WHILE file does OPEN

        DEFINE “,” as the separator of each parameter in each line of data

        IF there is LESS THAN OR EQUAL TO 1 “,” per line

            PRINT “File has less than two parameters per line. Please reformat and reload.”

            BREAK loop

```
        END IF
        IF TWO OR MORE “,” exists
            PUSH_BACK data before first “,” for each line into courseNumber vector
            PUSH_BACK data after first “,” for each line into courseName vector
            PUSH_BACK data after second “,” for each line into coursePrereq vector
                IF there are more “,” after the second “,”
                    PUSH_BACK data starting from the end of each line and
                    moving towards the front of the line
                END IF
            SEARCH courseNumber vector until there is a match with coursePrereq
            data from each line
                IF there is NO match
                    PRINT “This prerequisite course does not exist. Please
                    update file and reload.”
                    BREAK loop
                END IF
        END IF
        IF end of file is reached
            PRINT “File read successfully.”
        END IF
        CLOSE file
END WHILE
```

```
    WHILE file does NOT open
        PRINT "File cannot be opened. Please check file format and reload."
        INPUT course file
    END WHILE
END function
CREATE function to search and print information for a specific course
    REQUEST user to enter a course number
    INPUT user's entry
    WHILE user's entry is not empty
        SEARCH for user input in courseNumber vector until there is a match
        IF there is a match
            PRINT course number, course name, and any course prerequisite
            information
        END IF
        ELSE
            PRINT "There is no match."
        END ELSE
        REQUEST user to enter another course number
        INPUT user's entry
    END WHILE
END function
CREATE Partition() function
```

DECLARE variable “low” with INTEGER data type

ASSIGN “low” variable to “begin” for the first element of the courseNumber vector

DECLARE variable “high” with INTEGER data type

ASSIGN “high” variable to “end” for the last element of the courseNumber vector

DECLARE variable “pivot” with STRING data type

ASSIGN “pivot” to the middle element of the courseNumber vector

WHILE the low index is LESS THAN the high index

    WHILE element assigned to the “low” index is LESS THAN element assigned to “pivot”

        Increment low index

    END WHILE

    WHILE element assigned to “pivot” is LESS THAN element assigned to the “high” index

        Decrement high index

    END WHILE

    IF low index is LESS THAN high index

        Swap the low and high elements

        Increment “low”

        Decrement “high”

    END IF

END WHILE

RETURN “high”

END Partition() function

CREATE QuickSort() function

DECLARE variable “mid” with INTEGER data type

ASSIGN “mid” to 0

IF “begin” is GREATER THAN OR EQUAL to “end”

RETURN

END IF

ASSIGN “mid” to the location of last element in “low”

Recursively sort low partition from “begin” to “mid”

Recursively sort high partition from “mid” plus 1 to “end”

END QuickSort() function

Complete Main() function to print the sorted list

CALL quickSort() function to PRINT in alphanumeric order by course number along with each course’s name and prerequisite

END Main() function

Hash Table Data Structure

START

INCLUDE FSTREAM library to open and read file

CREATE course STRUCTURE

DECLARE courseNumber, courseName, and coursePrereq variables with STRING data type

END STRUCTURE

CREATE "HashTable” class function

DECLARE PRIVATE data members

DEFINE NODE STRUCTURE to hold course information

DECLARE default constructor

INITIALIZE default constructor with a course

INITIALIZE default constructor with a course and a key

END NODE STRUCTURE

DECLARE “courses” VECTOR of NODE type

DECLARE hash table’s key

DECLARE PUBLIC data members

END “HashTable” class function

CREATE function to load, read, and parse course file

DECLARE variable line

CREATE object of ifstream class to OPEN course file

WHILE file does OPEN

DEFINE “,” as the separator of each parameter in each line of data

ASSIGN data before first “,” for each line into “courseNumber”

ASSIGN data after first “,” for each line into “courseName”

ASSIGN data after second “,” for each line into “coursePrereq”

```
                IF there are more "," after the second ","
                    ASSIGN data starting from the end of each line and moving
                    towards the front of the line
                END IF
            SEARCH "courseNumber" until there is a match with "coursePrereq" data from
            each line
                IF there is NO match
                    PRINT "This prerequisite course does not exist. Please update file
                    and reload."
                    BREAK loop
                END IF
            IF end of file is reached
                PRINT "File read successfully."
            END IF
            CLOSE file
        END WHILE
        WHILE file does NOT open
            PRINT "File cannot be opened. Please check file format and reload."
            INPUT course file
        END WHILE
END function
CREATE INSERT function
```

```
        Use HASH function to get the key
        Retrieve “oldNode” NODE using the key
        IF no entry matches the “key”
                ASSIGN new node to the key position
        END IF
        ELSE
                IF node is not used
                        ASSIGN “oldNode” key to UINT_MAX
                        SET “oldNode” key to “key”
                        SET “oldNode” course to “course”
                        SET “oldNode” next to NULL
                END IF
                ELSE there is a collision
                        WHILE “oldNode” next is NOT NULL
                                ASSIGN “oldNode” to next open node
                        END WHILE
                        ADD new node to the end
                END ELSE
        END ELSE
END function
CREATE function to search and print information for a specific course
        REQUEST user to enter a course number
```

```
        INPUT user’s entry
        WHILE user’s entry is not empty
                SEARCH for user input using courseNumber until there is a match
                IF there is a match
                        PRINT course number, course name, and any course prerequisite
                        information
                END IF
                ELSE
                        PRINT “There is no match.”
                END ELSE
                REQUEST user to enter another course number
                INPUT user’s entry
        END WHILE
END function
CREATE PrintList() function
        DECLARE “iterator” with AUTO data type
        FOR when “iterator" does NOT equal the end of “nodes”
                ASSIGN “iterator” to the beginning of “nodes”
                IF the iterator’s key is NOT empty
                        PRINT “Key : ”, the iterator’s key, course number, course name, and
                        course prerequisite
                        PRINT new line
```

```
                ASSIGN node to the next iterator
                WHILE node is NOT NULL
                    PRINT “Key : ”, the node’s key, course number, course name,
                    and course prerequisite
                    PRINT new line
                    ASSIGN “node” to the next node
                END WHILE
            END IF
            INCREASE “iterator”
        END FOR
END function
```

Tree Data Structure

```
START
INCLUDE FSTREAM library to open and read file
CREATE course STRUCTURE to hold course information
    DECLARE “courseNumber” variable with STRING data type
    DECLARE “courseName” variable with STRING data type
    DECLARE “coursePrereq” variable with STRING data type
END STRUCTURE
CREATE internal STRUCTURE for tree node
    DECLARE course variable with Course structure
```

DECLARE left node

DECLARE right node

CREATE default constructor

ASSIGN left node to NULL

ASSIGN right node to NULL

END default constructor

INITIALIZE default constructor with a course

END internal STRUCTURE

CREATE “BinarySearchTree” class function

DECLARE PRIVATE data members and methods

DECLARE Node* root

DECLARE void addNode(Node* node, Course course)

DECLARE void inOrder(Node* node)

DECLARE void postOrder(Node* node)

DECLARE void preOrder(Node* node)

DECLARE Node* removeNode(Node* node, string courseNumber)

DECLARE PUBLIC data members and methods

BinarySearchTree()

virtual ~BinarySearchTree()

void InOrder()

void PostOrder()

void PreOrder()

```
        void Insert(Course course)
        void Remove(string courseNumber)
        Search(string courseNumber)
END “BinarySearchTree” class function
CREATE function to load, read, and parse course file
    DECLARE variable line
    CREATE object of ifstream class to OPEN course file
    WHILE file does OPEN
        DEFINE “,” as the separator of each parameter in each line of data
        ASSIGN data before first “,” for each line into “courseNumber”
        ASSIGN data after first “,” for each line into “courseName”
        ASSIGN data after second “,” for each line into “coursePrereq”
            IF there are more “,” after the second “,”
                ASSIGN data starting from the end of each line and moving
                towards the front of the line
            END IF
        SEARCH “courseNumber” until there is a match with “coursePrereq” data from
        each line
            IF there is NO match
                PRINT “This prerequisite course does not exist. Please update file
                and reload.”
                BREAK loop
```

```
                END IF
            IF end of file is reached
                PRINT “File read successfully.”
            END IF
            CLOSE file
        END WHILE
        WHILE file does NOT open
            PRINT “File cannot be opened. Please check file format and reload.”
            INPUT course file
        END WHILE
END function
CREATE INSERT function
        IF root EQUALS NULL
            ASSIGN root to NEW Node(course)
        END IF
        ELSE
            CALL addNode() function to add course at root level
        END ELSE
END function
CREATE addNote() function
        IF node is NOT empty AND larger
            IF there is no left node
```

```
                ASSIGN NEW Node(course) to left node
                RETURN
            END IF
            ELSE
                Recurse down the left node
            END ELSE
        END IF
        ELSE IF node is NOT empty AND smaller
            IF there is no right node
                ASSIGN NEW Node(course) to right node
                RETURN
            END IF
            ELSE
                Recurse down the right node
            END ELSE
        END ELSE IF
END addNote() function
CREATE function to search and print information for a specific course
        REQUEST user to enter a course number
        INPUT user's entry
        WHILE user's entry is not empty
            SEARCH for user input using courseNumber until there is a match
```

```
            IF there is a match
                PRINT course number, course name, and any course prerequisite
                information
            END IF
            ELSE
                PRINT "There is no match."
            END ELSE
            REQUEST user to enter another course number
            INPUT user's entry
        END WHILE
END function
CREATE InOrder() function
        CALL inOrder(root) function
END function
CREATE inOrder(Node* node) function
        IF node is NOT NULL
            CALL inOrder() function and pass in left node
            PRINT node with course's number, name, and prerequisites
            CALL inOrder() function and pass in right node
        END IF
END function
```

Menu

START by CREATING userMenu() function

DECLARE “userInput” variable with INTEGER data type

DISPLAY “User Menu” menu that includes “1. Load Data Structure”, “2. Print Course List”, “3. Print Course”, and “4. Exit”

PRINT message to request user to “Enter a number from the menu to continue”

INPUT user’s entry

WHILE user’s entry is NOT BETWEEN 1 AND 4

DISPLAY “User Menu” menu again

PRINT message “Please enter a valid number from the menu to continue”

INPUT user’s entry

END WHILE

WHILE user’s entry is BETWEEN 1 AND 4

IF user does NOT enter 4

IF user enters 1

CALL loadData() function to LOAD “Course Information” file into data structure

END IF

IF user enters 2

CALL printList() function to PRINT alphanumerically ordered list of all courses in the Computer Science department

END IF

```
            IF user enters 3
                CALL printCourse() function to PRINT title and prerequisites for
                any individual course
            END IF
            DISPLAY "User Menu" menu again
            PRINT message to request user to "Enter a number from the menu to
            continue"
        END IF
        ELSE
            DISPLAY "Goodbye" message
            BREAK
        END ELSE
    END WHILE
END function
```

Evaluation

Vector Data Structure

Code	Line Cost	# Times Executes	Total Cost
INCLUDE FSTREAM library to open and read file	1	1	1
ADD STRING library	1	1	1
CREATE course STRUCTURE	N/A	N/A	N/A

Code	Line Cost	# Times Executes	Total Cost
DECLARE courseNumber, courseName, and coursePrereq vectors with STRING data type	1	n	n
DEFINE all three above vectors to PUSH_BACK user input	1	n	n
DEFINE all three above vectors to separate data in each vector using “,”	1	n	n
END STRUCTURE	N/A	N/A	N/A
CREATE function to load, read, and parse course file	N/A	N/A	N/A
DECLARE variable line	1	1	1
CREATE object of ifstream class to OPEN course file	1	1	1
WHILE file does OPEN	1	n	n
DEFINE “,” as the separator of each parameter in each line of data	1	n	n
IF there is LESS THAN OR EQUAL TO 1 “,” per line	1	n	n
PRINT “File has less than two parameters per line. Please reformat and reload.”	1	n	n
BREAK loop	1	n	n
END IF	N/A	N/A	N/A
IF TWO OR MORE “,” exists	1	n	n

Code	Line Cost	# Times Executes	Total Cost
PUSH_BACK data before first “,” for each line into courseNumber vector	1	n	n
PUSH_BACK data after first “,” for each line into courseName vector	1	n	n
PUSH_BACK data after second “,” for each line into coursePrereq vector	1	n	n
IF there are more “,” after the second “,”	1	n	n
PUSH_BACK data starting from the end of each line and moving towards the front of the line	1	n	n
END IF	N/A	N/A	N/A
SEARCH courseNumber vector until there is a match with coursePrereq data from each line	1	n	n
IF there is NO match	1	n	n
PRINT “This prerequisite course does not exist. Please update file and reload.”	1	n	n
BREAK loop	1	n	n
END IF	N/A	N/A	N/A
END IF	N/A	N/A	N/A
IF end of file is reached	1	n	n

Code	Line Cost	# Times Executes	Total Cost
PRINT “File read successfully.”	1	n	n
END IF	1	n	n
CLOSE file	1	n	n
END WHILE	N/A	N/A	N/A
WHILE file does NOT open	1	n	n
PRINT “File cannot be opened. Please check file format and reload.”	1	n	n
INPUT course file	1	n	n
END WHILE	N/A	N/A	N/A
END function	N/A	N/A	N/A
		Total Cost	25n + 4
		Runtime	O(n)

Hash Table Data Structure

Code	Line Cost	# Times Executes	Total Cost
INCLUDE FSTREAM library to open and read file	1	1	1
CREATE course STRUCTURE	N/A	N/A	N/A
DECLARE courseNumber, courseName, and coursePrereq variables with STRING data type	1	n	n
END STRUCTURE	N/A	N/A	N/A
CREATE “HashTable” class function	N/A	N/A	N/A
DECLARE PRIVATE data members	1	n	n

Code	Line Cost	# Times Executes	Total Cost
DEFINE NODE STRUCTURE to hold course information	1	1	1
DECLARE default constructor	1	1	1
INITIALIZE default constructor with a course	1	1	1
INITIALIZE default constructor with a course and a key	1	1	1
END NODE STRUCTURE	N/A	N/A	N/A
DECLARE “courses” VECTOR of NODE type	1	1	1
DECLARE hash table’s key	1	1	1
DECLARE PUBLIC data members	1	n	n
END “HashTable” class function	N/A	N/A	N/A
CREATE function to load, read, and parse course file	N/A	N/A	N/A
DECLARE variable line	1	1	1
CREATE object of ifstream class to OPEN course file	1	1	1
WHILE file does OPEN	1	n	n
DEFINE “,” as the separator of each parameter in each line of data	1	n	n
ASSIGN data before first “,” for each line into “courseNumber”	1	n	n
ASSIGN data after first “,” for each line into “courseName”	1	n	n

Code	Line Cost	# Times Executes	Total Cost
ASSIGN data after second “,” for each line into “coursePrereq”	1	n	n
IF there are more “,” after the second “,”	1	n	n
ASSIGN data starting from the end of each line and moving towards the front of the line	1	n	n
END IF	N/A	N/A	N/A
SEARCH “courseNumber” until there is a match with “coursePrereq” data from each line	1	n	n
IF there is NO match	1	n	n
PRINT “This prerequisite course does not exist. Please update file and reload.”	1	n	n
BREAK loop	1	n	n
END IF	N/A	N/A	N/A
IF end of file is reached	1	n	n
PRINT “File read successfully.”	1	n	n
END IF	N/A	N/A	N/A
CLOSE file	1	n	n
END WHILE	N/A	N/A	N/A
WHILE file does NOT open	1	n	n
PRINT “File cannot be opened. Please check file format and reload.” INPUT course file	1	n	n
END WHILE	N/A	N/A	N/A
CREATE INSERT function	N/A	N/A	N/A
Use HASH function to get the key	1	1	1

Code	Line Cost	# Times Executes	Total Cost
Retrieve “oldNode” NODE using the key	1	1	1
IF no entry matches the “key”	1	1	1
ASSIGN new node to the key position	1	1	1
END IF	N/A	N/A	N/A
ELSE	1	1	1
IF node is not used	1	1	1
ASSIGN “oldNode” key to UINT_MAX	1	1	1
SET “oldNode” key to “key”	1	1	1
SET “oldNode” course to “course”	1	1	1
SET “oldNode” next to NULL	1	1	1
END IF	N/A	N/A	N/A
ELSE there is a collision	1	1	1
WHILE “oldNode” next is NOT NULL	1	n	n
ASSIGN “oldNode” to next open node	1	n	n
END WHILE	N/A	N/A	N/A
ADD new node to the end	1	1	1
END ELSE	N/A	N/A	N/A
END ELSE	N/A	N/A	N/A
END function	N/A	N/A	N/A
		Total Cost	21n + 21

Code	Line Cost	# Times Executes	Total Cost
		Runtime	O(n)

Tree Data Structure

Code	Line Cost	# Times Executes	Total Cost
INCLUDE FSTREAM library to open and read file	1	1	1
CREATE course STRUCTURE to hold course information	N/A	N/A	N/A
DECLARE “courseNumber” variable with STRING data type	1	1	1
DECLARE “courseName” variable with STRING data type	1	1	1
DECLARE “coursePrereq” variable with STRING data type	1	1	1
END STRUCTURE	N/A	N/A	N/A
CREATE internal STRUCTURE for tree node	N/A	N/A	N/A
DECLARE course variable with Course structure	1	1	1
DECLARE left node	1	1	1
DECLARE right node	1	1	1
CREATE default constructor	N/A	N/A	N/A
ASSIGN left node to NULL	1	1	1
ASSIGN right node to NULL	1	1	1

Code	Line Cost	# Times Executes	Total Cost
END default constructor	N/A	N/A	N/A
INITIALIZE default constructor with a course	1	1	1
END internal STRUCTURE	N/A	N/A	N/A
CREATE “BinarySearchTree” class function	N/A	N/A	N/A
DECLARE PRIVATE data members and methods	N/A	N/A	N/A
DECLARE Node* root	1	1	1
DECLARE void addNode(Node* node, Course course)	1	1	1
DECLARE void inOrder(Node* node)	1	1	1
DECLARE void postOrder(Node* node)	1	1	1
DECLARE void preOrder(Node* node)	1	1	1
DECLARE Node* removeNode(Node* node, string courseNumber)	1	1	1
DECLARE PUBLIC data members and methods	N/A	N/A	N/A
BinarySearchTree()	1	1	1
virtual ~BinarySearchTree()	1	1	1
void InOrder()	1	1	1
void PostOrder()	1	1	1
void PreOrder()	1	1	1
void Insert(Course course)	1	1	1
void Remove(string courseNumber)	1	1	1
Bid Search(string courseNumber)	1	1	1

Code	Line Cost	# Times Executes	Total Cost
END “BinarySearchTree” class function	N/A	N/A	N/A
CREATE function to load, read, and parse course file	N/A	N/A	N/A
DECLARE variable line	1	1	1
CREATE object of ifstream class to OPEN course file	1	1	1
WHILE file does OPEN	1	n	n
DEFINE “,” as the separator of each parameter in each line of data	1	n	n
ASSIGN data before first “,” for each line into “courseNumber”	1	n	n
ASSIGN data after first “,” for each line into “courseName”	1	n	n
ASSIGN data after second “,” for each line into “coursePrereq”	1	n	n
IF there are more “,” after the second “,”	1	n	n
ASSIGN data starting from the end of each line and moving towards the front of the line	1	n	n
END IF	N/A	N/A	N/A
SEARCH “courseNumber” until there is a match with “coursePrereq” data from each line	1	n	n
IF there is NO match	1	n	n
PRINT “This prerequisite course does not exist. Please update file and reload.”	1	n	n

Code	Line Cost	# Times Executes	Total Cost
BREAK loop	1	n	n
END IF	N/A	N/A	N/A
IF end of file is reached	1	n	n
PRINT “File read successfully.”	1	n	n
END IF	N/A	N/A	N/A
CLOSE file	1	n	n
END WHILE	N/A	N/A	N/A
WHILE file does NOT open	1	n	n
PRINT “File cannot be opened. Please check file format and reload.”	1	n	n
INPUT course file	1	n	n
END WHILE	N/A	N/A	N/A
END function	N/A	N/A	N/A
CREATE INSERT function	N/A	N/A	N/A
IF root EQUALS NULL	1	1	1
ASSIGN root to NEW Node(course)	1	1	1
END IF	N/A	N/A	N/A
ELSE	1	1	1
CALL addNode() function to add course at root level	1	1	1
END ELSE	N/A	N/A	N/A
END function	N/A	N/A	N/A
CREATE addNote() function	N/A	N/A	N/A
IF node is NOT empty AND larger	1	1	1
IF there is no left node	1	1	1
ASSIGN NEW Node(course) to left node	1	1	1
RETURN	1	1	1
END IF	N/A	N/A	N/A

Code	Line Cost	# Times Executes	Total Cost
ELSE	1	1	1
Recurse down the left node	1	1	1
END ELSE	N/A	N/A	N/A
END IF	N/A	N/A	N/A
ELSE IF node is NOT empty AND smaller	1	1	1
IF there is no right node	1	1	1
ASSIGN NEW Node(course) to right node	1	1	1
RETURN	1	1	1
END IF	N/A	N/A	N/A
ELSE	1	1	1
Recurse down the right node	1	1	1
END ELSE	N/A	N/A	N/A
END ELSE IF	N/A	N/A	N/A
END addNote() function	N/A	N/A	N/A
		Total Cost	17n + 42
		Runtime	O(n)

Advantages vs. Disadvantages

All of the evaluations below are based on the advisors' requirements of:

1. Print a list of all the Computer Science courses in alphanumeric order. This use case requires the program to be able to access the data, sort, and then print the data.
2. For a given course, print out its title and prerequisites. This use case requires the program to be able to access the data, search for the specific course, and then print the data affiliated with that course.

Vector Data Structure

The advantages are:

- It has both a worst and average-time complexity of O(1) for accessing data.
- The operation is simple and straightforward to use on small data sets such as the data in the given Course Information file.
- The data do not need to be arranged beforehand.

The disadvantages are:

- It has both a worst and average-time complexity of O(n) for searching for data.
- It has a worst-space complexity of O(n).
- It is not ideal for the second requirement mentioned above, since it has to search through each data one-by-one using a linear or sequential approach before it can print the information associated with that data point. Furthermore, and thinking ahead, this means that this data structure does not scale well, such as to support a file with more courses or data points. There is a higher computational cost as the size of the data set grows.

Hash Table Data Structure

The advantages are:

- The average-time complexity for searching for data is O(1).
- It is more efficient to search through large data sets, because of its divide and conquer approach and two-way traversal capability. Thinking ahead, this means that this data structure is scalable to support a growing list of courses or data points.

The disadvantages are:

- The worst-time complexity for searching for data is O(n).
- It has a worst-space complexity of O(n).
- The data are stored in an unordered manner, which means that it takes more efforts to complete the first use case mentioned above.
- A key is required to access the value in a hash table. The key has to be an integer data type, which means that the courses have to be inserted as an integer data type, and then the letters that are a part of the course identification number have to be removed before the course number can be used as the key. This adds additional steps and run time.
- Collision can occur when inserting new data points into an existing table.

Tree Data Structure

The advantages are:

- The average time complexity for accessing and searching for data is O(log(n)).
- An inorder traversal algorithm allows the ability to visit all nodes in a binary search tree (BST) from smallest to largest, making it easier and faster to accomplish the first use case mentioned above.
- The natural ordering of a BST makes it easy and fast to detect a node's predecessor and successor, which makes it easier and faster to find a course's prerequisite and complete the second use case mentioned above.

The disadvantages are:

- The worst-case space complexity is O(n).
- A BST has to be balanced on either side of the tree in order to reduce operation costs that could turn searches into a linear approach similar to an array.

Recommendation

I plan to utilize the BST in my code. The BST has a constant O(log(n)) average time complexity for accessing and searching for data, which are some of the main operations that are required to meet the program's use cases. Furthermore, the default sorting structure of smallest to largest makes the BST a better data structure to accomplish the advisors' need to print a list of all the Computer Science courses in alphanumeric order. This is especially true when compared to conducting this operation using a vector or hash table, which both require additional operations and run time to sort data in alphanumeric order.

Moreover, the advisors' need to search for a specific course and then print the title and prerequisites of that course requires a data structure that can support fast access and search operations. Vectors have both an average and worst time complexities of O(1) for accessing data, which is better than the O(log(n)) average and O(n) worst time complexities that the BST has for this operation. However, vectors also have an average and worst time complexities of O(n) for searching (and inserting and deleting) data. This makes a vector not the best data structure of choice when compared to the O(log(n)) average and O(n) worst time complexities that a BST has for the same operation(s).

A hash table is also not the most time efficient data structure to meet the program's use cases, primarily due to the requirements of a hash key and function and presence of collisions. The hash key must be an integer data type, which is a requirement that does not work well with the provided Course Information file because of the string data type of all of the data in the file. This means that additional operations and run time must take place to convert the string data type into an integer data type before the hash key can be created and function can be performed. The consistency of an average time complexity of O(log(n)) for accessing, searching, inserting, and deleting data of a BST makes it a better choice than a hash table, even if both data structures have a worst time complexity of O(n).

Last, but not least, the default ordering of a BST makes it easier to identify predecessors and successors of a node or course. This makes it faster to detect a course's prerequisite than a vector or hash table data structure. In turn and conclusion, this allows the BST to be a better data structure to support the advisors' need to identify and print a specific course's title and prerequisites.