Vector pseudocode:

// Defines course structure

Structure Course

String courseNumber

String courseTitle

List of Strings prerequisites

// Creates course object

Function CreateCourseFromLine(line)

// Split the line by commas to extract the course details

Split line into parts by comma delimiter and store in array 'tokens'

// Creates new course

Declare Course course

// Assigns course number and title

course.courseNumber = tokens[0]

course.courseTitle = tokens[1]

// Checks for prerequisites

For i = 2 to length of tokens - 1

Add tokens[i] to course.prerequisites

Return course

// Loads courses from file into the vector

Function LoadCourses(filename)

Declare Vector of Course courses // The vector to store all course objects

Open file with name 'filename' for reading

If file is not open

Print "Error: Unable to open file."

Return empty vector

// Reads file

While not end of file

Read line from file

// Skips any empty lines

If line is empty

Continue to the next iteration

// Creates course from lines

Declare Course course = CreateCourseFromLine(line)

Add course to courses

Close file

// Checks if missing any prerequisites

For each course in courses

For each prerequisite in course.prerequisites

If prerequisite is not found in the list of course.courseNumbers in courses

Print "Prerequisite not met."

Return courses

// Searches courses by courseNumber

Function FindCourseByNumber(courses, targetCourseNumber)

For each course in courses

If course.courseNumber == targetCourseNumber

Return course

// If no course found with number,

Return null

// Displays course info

Function DisplayCourseInformation(course)

If course is null

Print "Course not found."

Return

Print "Course Number: " + course.courseNumber

Print "Course Title: " + course.courseTitle

If course.prerequisites is not empty

Print "Prerequisites: " + Join(course.prerequisites, ", ")

Else

Print "Prerequisites: None"

// Displays all courses

Function DisplayCourses(courses)

If courses is empty

Print "No courses available."

Return

For each course in courses

DisplayCourseInformation(course)

Hash table Pseudocode:  
  
// Defines course structure

Structure Course

String courseNumber

String courseTitle

List of Strings prerequisites

// Creates course object

Function CreateCourseFromLine(line)

// Split the line by commas to extract the course details

Split line into parts by comma delimiter and store in array 'tokens'

// Validate the line format with course number and title

If length of tokens < 2

Print "Error: Invalid format in line - " + line

Return null

// Creates new course

Declare Course course

// Assigns course number and title

course.courseNumber = tokens[0]

course.courseTitle = tokens[1]

// Checks for prerequisites

For i = 2 to length of tokens - 1

Add tokens[i] to course.prerequisites

Return course

// Loads courses from file into both a vector and a hash table

Function LoadCourses(filename)

Declare Vector of Course courses // The vector to store all course objects

Declare HashTable of Course courseTable // The hash table to store course objects, courseNumber is key

Open file with name 'filename' for reading

If file is not open

Print "Error: Unable to open file."

Return empty vector and hash table

// Reads file

While not end of file

Read line from file

// Skips any empty lines

If line is empty

Continue to the next iteration

// Creates course from line

Declare Course course = CreateCourseFromLine(line)

// If the course is null due to format error, skip it

If course is null

Continue

// Add course to both data structures

Add course to courses

Insert course into courseTable with key course.courseNumber

Close file

// Checks if any prerequisites are missing

For each course in courses

For each prerequisite in course.prerequisites

If prerequisite is not found in courseTable

Print "Error: Prerequisite " + prerequisite + " for course " + course.courseNumber + " not found."

Return courses, courseTable

// Searches for a course by courseNumber in the vector and hash table

Function FindCourseByNumber(courses, courseTable, targetCourseNumber)

// First try to find in the hash table (more efficient)

If targetCourseNumber exists in courseTable

Return courseTable[targetCourseNumber]

// If not found in the hash table, iterate through the vector (backup check)

For each course in courses

If course.courseNumber == targetCourseNumber

Return course

// If no course found with the number

Return null

// Displays course info

Function DisplayCourseInformation(course)

If course is null

Print "Course not found."

Return

Print "Course Number: " + course.courseNumber

Print "Course Title: " + course.courseTitle

If course.prerequisites is not empty

Print "Prerequisites: " + Join(course.prerequisites, ", ")

Else

Print "Prerequisites: None"

// Displays all courses

Function DisplayCourses(courses)

If courses is empty

Print "No courses available."

Return

For each course in courses

DisplayCourseInformation(course)

Print a blank line

// Main function to execute the program

Function Main

Declare String filename = "courseData.txt" // The name of the file containing course data

// Load courses from the file into both the vector and hash table

Declare Vector of Course courses

Declare HashTable of Course courseTable

(courses, courseTable) = LoadCourses(filename)

// Display all courses using vector

Print "All Courses:"

DisplayCourses(courses)

// Get user input for the course they want to search for

Print "Enter the course number you want to search for (e.g., CSCI300): "

Read userInput

// Search for the course using both vector and hash table

Declare Course foundCourse = FindCourseByNumber(courses, courseTable, userInput)

// Display the found course's information

DisplayCourseInformation(foundCourse)

// Call the main function to execute the program

Main

Tree Pseudocode:

// Defines course structure

Structure Course

String courseNumber

String courseTitle

List of Strings prerequisites

// Defines the BST node structure

Structure TreeNode

Course course

TreeNode leftChild

TreeNode rightChild

// Creates a course from given line

Function CreateCourseFromLine(line)

// Split the line by commas to extract the course details

Split line into parts by comma delimiter and store in array 'tokens'

// Validates the line format (at least course number and title required)

If length of tokens < 2

Print "Error: Invalid format in line - " + line

Return null

// Creates new course

Declare Course course

course.courseNumber = tokens[0]

course.courseTitle = tokens[1]

// Adds prerequisites if there are any

For i = 2 to length of tokens - 1

Add tokens[i] to course.prerequisites

Return course

// Inserts course into BST by course number

Function InsertIntoBST(root, newCourse)

If root is null

Declare TreeNode newNode

newNode.course = newCourse

newNode.leftChild = null

newNode.rightChild = null

Return newNode

Else If newCourse.courseNumber < root.course.courseNumber

root.leftChild = InsertIntoBST(root.leftChild, newCourse)

Else

root.rightChild = InsertIntoBST(root.rightChild, newCourse)

Return root

// Searches for course by course number

Function SearchBST(root, targetCourseNumber)

If root is null

Return null

Else If root.course.courseNumber == targetCourseNumber

Return root.course

Else If targetCourseNumber < root.course.courseNumber

Return SearchBST(root.leftChild, targetCourseNumber)

Else

Return SearchBST(root.rightChild, targetCourseNumber)

// Loads courses from file then stores them in the BST

Function LoadCourses(filename)

Declare TreeNode root = nul

Open file with name 'filename' for reading

If file is not open

Print "Error: Unable to open file."

Return root

// Reads file

While not end of file

Read line from file

If line is empty

Continue

// Creates course from line

Declare Course course = CreateCourseFromLine(line)

// Inserts course into the BST

root = InsertIntoBST(root, course)

Close file

// Checks if prerequisites are valid

ValidatePrerequisites(root)

Return root

// Validates prerequisites by checking that each prerequisite exists in the BST

Function ValidatePrerequisites(root)

// In-order traversal of the BST to check prerequisites

If root is not null

ValidatePrerequisites(root.leftChild)

// For each prerequisite of the current course

For each prerequisite in root.course.prerequisites

If SearchBST(root, prerequisite) is null

Print "Error: Prerequisite " + prerequisite + " for course " + root.course.courseNumber + " not found."

ValidatePrerequisites(root.rightChild)

// Displays course info

Function DisplayCourseInformation(course)

If course is null

Print "Course not found."

Return

Print "Course Number: " + course.courseNumber

Print "Course Title: " + course.courseTitle

If course.prerequisites is not empty

Print "Prerequisites: " + Join(course.prerequisites, ", ")

Else

Print "Prerequisites: None"

// Traverses BST in-order

Function DisplayAllCourses(root)

If root is not null

DisplayAllCourses(root.leftChild)

DisplayCourseInformation(root.course)

Print a blank line

DisplayAllCourses(root.rightChild)

// Main()

Function Main

Declare String filename = "courseData.txt" // The name of the file containing course data

// Loads courses from the file into BST

Declare TreeNode root = LoadCourses(filename)

// Display all courses in the binary search tree

Print "All Courses:"

DisplayAllCourses(root)

// Gets user input to search

Print "Enter course number to searc:"

Read userInput

// Searches for the course with BST

Declare Course foundCourse = SearchBST(root, userInput)

// Displays thecourse's information

DisplayCourseInformation(foundCourse)

// Calls the main()

Main

Menu Pseudocode:

// Displays menu choices

Function DisplayMenu

Print "1: Load course data"

Print "2: Print sorted courses"

Print "3: Print a course's information"

Print "9: Exit"

// Handles user choices

Function MainMenu

Declare Boolean isDataLoaded = false

Declare Vector of Course courses // Stores courses in vector,

Declare HashTable of Course courseTable // Hash table,

Declare TreeNode root // and BST

Declare Integer choice

// Loops to keep menu visible

While true

DisplayMenu()

Read choice

// Menu option switch cases

Switch choice

Case 1:

// Load course data into data structures

Print "Loading course data..."

Declare String filename = "courseData.txt"

(courses, courseTable) = LoadCourses(filename)

root = LoadCoursesIntoBST(filename)

If courses is not empty

Print "Course data loaded successfully."

isDataLoaded = true

Else

Print "Error: Failed to load course data."

Case 2:

// Print courses alphanumerically

If isDataLoaded == false

Print "Error, not loaded"

Else

Print "All courses (sorted alphabetically by course number):"

DisplayAllCourses(root) // Traverses BST to print courses

Case 3:

// Prints details for a given course

If isDataLoaded == false

Print "Error, not loaded"

Else

Print "Enter the course number (e.g., CSCI300):"

Declare String courseNumber

Read courseNumber

// Searches BST and displays course information

Declare Course foundCourse = SearchBST(root, courseNumber)

If foundCourse is null

Print "Error: Course not found"

Else

DisplayCourseInformation(foundCourse)

Case 9:

Exit

Default:

Print "Invalid choice, choose an option from the menu.."

Function Main

// Calls menu function

MainMenu()

Main

Print sorted Pseudocode:

Vector:

// Sorts courses by number

Function SortCoursesByNumber(courses)

// Sorts vector by course number

Sort courses by courseNumber in ascending order

Return courses

// Prints sorted courses from vector

Function DisplaySortedCoursesVector(courses)

If courses is empty

Print "No courses available"

Return

// Sorts courses by course number

Declare Vector sortedCourses = SortCoursesByNumber(courses)

// Prints sorted courses

For each course in sortedCourses

Print "Course Number: " + course.courseNumber + " - Title: " + course.courseTitle

If course.prerequisites is not empty

Print " Prerequisites: " + Join(course.prerequisites, ", ")

Else

Print " Prerequisites: None"

Print a blank line

Hash Table:

// Extracts courses from hash table, into vector

Function ExtractCoursesFromHashTable(courseTable)

Declare Vector courseList

For each course in courseTable

Add course to courseList

Return courseList

// Displays sorted course with hash table

Function DisplaySortedCoursesHashTable(courseTable)

// Extracts courses into vector

Declare Vector courseList = ExtractCoursesFromHashTable(courseTable)

// Re-use vector sorting

DisplaySortedCoursesVector(courseList)

BST:

// Prints sorted courses with in-order traversal

Function DisplayAllCoursesBST(root)

// Traverses in-order

If root is not null

// Traverses left

DisplayAllCoursesBST(root.leftChild)

// Prints current course

Print "Course Number: " + root.course.courseNumber + " - Title: " + root.course.courseTitle

If root.course.prerequisites is not empty

Print " Prerequisites: " + Join(root.course.prerequisites, ", ")

Else

Print " Prerequisites: None"

Print a blank line

// Traverses right

DisplayAllCoursesBST(root.rightChild)

Runtime Analyses:

| Code | Line Cost | #Times Executes | Total Cost |
| --- | --- | --- | --- |
| For each Line | 1 | n | n |
| If line is empty, skip | 1 | n | n |
| Splitting lines at comma | 1 | n | n |
| Error if not formatted correctly | 1 | n | n |
| Assigning course number and title | 1 | n | n |
| For each prerequisite | 1 | n\*p | n\*p |
| Add prerequisite | 1 | n\*p | n\*p |
| Add course to vector | 1 | n | n |
| Add course to hash table | 1 | n | n |
| Add course to BST | log(n) | n | n\*log(n) |
| **Total** |  |  | 6n+2np+n\*log(n) |
| **Runtime** |  |  | O(n\*log(n)+n\*p) |

N is the number of courses, P is the number of prerequisites for the course.

Vectors have the advantage of being probably the simplest structure to implement, and are fairly efficient when displaying the entire course catalog, but their biggest advantage is probably their efficiency at having things inserted at the end. Their disadvantages are usually that they are typically not great for linear searches with large data sets, being relatively inefficient at sorting, and being poor at handling non-end insertions. A vector probably isn’t the most effective option, since it’s more geared toward smaller sets of data, and use cases where being resized frequently is expected.

Hash tables, assuming the function is good and the collision rate is low, have usually a fairly fast access time, and can more quickly access things like prerequisites. However, if the collision rate is high, that advantage goes away almost completely. Hash tables also are more complicated to print in a given order, since they don’t maintain the elements’ order, so sorting them is a relatively time-consuming process.

Binary search trees have the advantage of being pretty much necessarily already sorted, and if the tree is balanced, insertions, deletions, and searches are all fairly fast. If the tree becomes unbalanced, though, it essentially becomes a standard linked list, and loses most of its advantages, and keeping it balanced can be a difficult task.

With these things in mind, at the moment, I plan to use a binary search tree for the actual code. The disadvantages of needing to maintain balance isn’t a huge concern, in this case, as the dataset is presumably going to be static, since it’s a school project. Additionally, something about BSTs just feels intuitive to me in a way I am not sure how to define.