Devin Wheeler  
6-2 Submit Project One

|  |  |  |  |
| --- | --- | --- | --- |
| **Operation** | **Vector** | **Hash Tabbe** | **BST** |
| **File Open** | O(1) | O(1) | O(1) |
| **Loop Through Lines** | O(n) | O(n) | O(n) |
| **Validation of Prerequisites** | O(n²) | O(k) | O(k log n) |
| **Insertion** | O(1) | O(1) (avg), O(n)(worst) | O(log n) |
| **Search** | O(n) | O(1) (avg), O(n)(worst) | O(log n) |
| **Sorting** | O(n log n) | N/A | N/A |
| **Print All Courses** | O(n) | O(n) | O(n) |
| **Total Runtime** | O(n²) | O(n) (avg),O(n²)(worst) | O(n log n) |

**Explanation:**

**Vector:**

Sequential operations lead to O(n) for most tasks.

Poor scalability due to O(n²) for validation and O(n log n) for sorting.

**Hash Table:**

Fast average-case performance (O(1)) but can at worst case take O(n).

Efficient for lookups and insertions.

**Binary Search Tree:**

Balanced trees ensure O(log n) performance for insertion, search, and validation.

Inherent sorting makes it great for ordered traversal.

Slower if the tree isn’t

**Recommendation:**

I recommend using the hash table for the advising program because it’s the fastest option on average for searching and inserting courses. While it might slow down in rare cases due to collisions, a good hash function can help prevent this. If we need the courses to stay sorted, the binary search tree is a great other option, as it is fast and keeps everything organized. The vector is okay for small datasets but becomes too slow as the data grows.

**function loadCoursesFromFile(fileName):**

open file with name fileName

if file cannot be opened:

print "Unable to open file."

return empty vector

create an empty **Struct** //vector, hashtsble , or BST

for each line in file:

split line into tokens using comma as separation point

if number of tokens < 2:

print "Invalid line format."

continue

courseNumber = tokens[0]

courseTitle = tokens[1]

prerequisites = tokens[2:]

for each prerequisite in prerequisites:

if prerequisite does not exist in any line in the file:

print "Prerequisite", prerequisite, "does not exist."

continue

course = createCourse(courseNumber, courseTitle, prerequisites)

add course to courses **Struct** //vector, hashtsble , or BST

close file

return courses

**struct Course:**

string courseNumber

string courseTitle

list<string> prerequisites , vector<string> prerequisites

**function createCourse(courseNumber, courseTitle, prerequisites):**

initialize a Course object

set course.courseNumber = courseNumber

set course.courseTitle = courseTitle

set course.prerequisites = prerequisites

return course

**(For Hash table)**

**function insert(hashTable, course):**

key = hash(course.courseNumber)

if hashTable[key] is empty:

add course to hashTable at key

else:

traverse linked list at hashTable[key] to find an empty node

append course to the end of the chain

**(For BST)**

**function insert(binarySearchTree, course):**

if binarySearchTree.root is null:

set binarySearchTree.root to new node containing course

else:

call AddNode(binarySearchTree.root, course)

**function AddNode(node, course):**

if course.courseNumber < node.course.courseNumber:

if node.left is null:

set node.left to new node containing course

else:

call AddNode(node.left, course)

else:

if node.right is null:

set node.right to new node containing course

else:

call AddNode(node.right, course)

**(For BST)**

**function printCoursesInOrder(node):**

if node is not null:

printCoursesInOrder(node.left)

print "Course Number:", node.course.courseNumber

print "Course Title:", node.course.courseTitle

printCoursesInOrder(node.right)

**(For Hash)  
function printCourse(hashTable, courseNumber):**

key = hash(courseNumber)

course = search(hashTable, courseNumber)

if course is null:

print "Course", courseNumber, "not found."

return

print "Course Number:", course.courseNumber

print "Course Title:", course.courseTitle

if course.prerequisites is empty:

print "No prerequisites."

else:

print "Prerequisites:"

for each prerequisite in course.prerequisites:

print prerequisite

**function sortHashMapAlphabetically(hashTable):**

create an empty list sortedKeys

for each key in hashTable:

add key to sortedKeys

sort sortedKeys in ascending order

for each key in sortedKeys:

course = hashTable[key]

print "Course Number:", course.courseNumber

print "Course Title:", course.courseTitle

if course.prerequisites is empty:

print "No prerequisites."

else:

print "Prerequisites:"

for each prerequisite in course.prerequisites:

print prerequisite

**(For Vector)  
function searchCourse(courses, courseNumber):**

for each course in courses:

if course.courseNumber == courseNumber:

print "Course Number:", course.courseNumber

print "Course Title:", course.courseTitle

if course.prerequisites is empty:

print "No prerequisites."

else:

print "Prerequisites:"

for each prerequisite in course.prerequisites:

print prerequisite

return

print "Course", courseNumber, "not found."

**function displayMenu():**

print "1. Load course data"

print "2. Display all courses"

print "3. Search for a course"

print "9. Exit"

**function menu():**

while true:

displayMenu()

choice = get user input

if choice == 1:

loadCoursesFromFile("courses.txt")

elif choice == 2:

printAllCourses(dataStructure)

elif choice == 3:

courseNumber = get user input

searchCourse(dataStructure, courseNumber)

elif choice == 9:

break

else:

print "Invalid choice."