Tyler Allsopp  
CS-300  
March 1,2025  
Project One

## **1. Pseudocode**

### **1.1. Common Pseudocode for All Data Structures**

**Open File and Read Data**

FUNCTION LoadData(fileName)  
 OPEN fileName AS inputFile  
 IF inputFile IS NULL THEN  
 PRINT "Error: Cannot open file."  
 RETURN  
 END IF  
  
 WHILE NOT EndOfFile(inputFile)  
 READ line FROM inputFile  
 SPLIT line INTO courseID, courseTitle, prerequisites  
 IF courseID IS NULL OR courseTitle IS NULL THEN  
 PRINT "Error: Invalid format."  
 CONTINUE  
 END IF  
 CALL CreateCourse(courseID, courseTitle, prerequisites)  
 END WHILE  
  
 CLOSE inputFile  
END FUNCTION

**Create Course Object**

FUNCTION CreateCourse(courseID, courseTitle, prerequisites)  
 COURSE = NEW Course  
 COURSE.id = courseID  
 COURSE.title = courseTitle  
 COURSE.prerequisites = SPLIT prerequisites BY ","  
 CALL InsertCourse(COURSE)  
END FUNCTION

### **1.2. Pseudocode for Vector**

**Insert Course into Vector**

FUNCTION InsertCourse(course)  
 APPEND course TO courseVector  
END FUNCTION

**Sort and Print Vector**

FUNCTION PrintAllCourses()  
 SORT courseVector BY courseID  
 FOR EACH course IN courseVector  
 PRINT course.id, course.title  
 END FOR  
END FUNCTION

**Print Specific Course and Prerequisites**

FUNCTION PrintCourse(courseID)  
 FOR EACH course IN courseVector  
 IF course.id EQUALS courseID THEN  
 PRINT course.title  
 PRINT "Prerequisites: ", course.prerequisites  
 RETURN  
 END IF  
 END FOR  
 PRINT "Course not found."  
END FUNCTION

### **1.3. Pseudocode for Hash Table**

**Insert Course into Hash Table**

FUNCTION InsertCourse(course)  
 index = Hash(course.id)  
 hashTable[index] = course  
END FUNCTION

**Print All Courses**

FUNCTION PrintAllCourses()  
 FOR EACH index IN hashTable  
 IF hashTable[index] IS NOT NULL THEN  
 PRINT hashTable[index].id, hashTable[index].title  
 END IF  
 END FOR  
END FUNCTION

**Print Specific Course and Prerequisites**

FUNCTION PrintCourse(courseID)  
 index = Hash(courseID)  
 course = hashTable[index]  
 IF course IS NOT NULL THEN  
 PRINT course.title  
 PRINT "Prerequisites: ", course.prerequisites  
 ELSE  
 PRINT "Course not found."  
 END IF  
END FUNCTION

### **1.4. Pseudocode for Binary Search Tree**

**Insert Course into Tree**

FUNCTION InsertCourse(course)  
 IF root IS NULL THEN  
 root = course  
 ELSE  
 CALL InsertNode(root, course)  
 END IF  
END FUNCTION  
  
FUNCTION InsertNode(node, course)  
 IF course.id < node.id THEN  
 IF node.left IS NULL THEN  
 node.left = course  
 ELSE  
 CALL InsertNode(node.left, course)  
 END IF  
 ELSE  
 IF node.right IS NULL THEN  
 node.right = course  
 ELSE  
 CALL InsertNode(node.right, course)  
 END IF  
 END IF  
END FUNCTION

**Print All Courses (In-Order Traversal)**

FUNCTION PrintAllCourses()  
 CALL InOrderTraversal(root)  
END FUNCTION  
  
FUNCTION InOrderTraversal(node)  
 IF node IS NOT NULL THEN  
 CALL InOrderTraversal(node.left)  
 PRINT node.id, node.title  
 CALL InOrderTraversal(node.right)  
 END IF  
END FUNCTION

**Print Specific Course and Prerequisites**

FUNCTION PrintCourse(courseID)  
 node = root  
 WHILE node IS NOT NULL  
 IF courseID EQUALS node.id THEN  
 PRINT node.title  
 PRINT "Prerequisites: ", node.prerequisites  
 RETURN  
 ELSE IF courseID < node.id THEN  
 node = node.left  
 ELSE  
 node = node.right  
 END IF  
 END WHILE  
 PRINT "Course not found."  
END FUNCTION

### **1.5. Menu Pseudocode**

FUNCTION DisplayMenu()  
 PRINT "1: Load Data"  
 PRINT "2: Print All Courses"  
 PRINT "3: Print Course and Prerequisites"  
 PRINT "9: Exit"  
 GET userInput  
 SWITCH userInput  
 CASE 1:  
 CALL LoadData("courses.txt")  
 CASE 2:  
 CALL PrintAllCourses()  
 CASE 3:  
 PRINT "Enter Course ID:"  
 GET courseID  
 CALL PrintCourse(courseID)  
 CASE 9:  
 PRINT "Exiting program."  
 DEFAULT:  
 PRINT "Invalid option."  
 END SWITCH  
END FUNCTION

## **2. Runtime Analysis (Big O Notation)**

|  |  |  |  |
| --- | --- | --- | --- |
| **Operation** | **Vector** | **Hash Table** | **Binary Search Tree** |
| Insertion | O(1) (amortized) | O(1) | O(log n) |
| Search (Find Course) | O(n) | O(1) | O(log n) |
| Print All Courses (Sorted) | O(n log n) | O(n log n) | O(n) |
| Space Complexity | O(n) | O(n) | O(n) |

## **3. Evaluation of Data Structures**

**Vector**

* **Advantages:** Simple to implement; efficient for sequential access.
* **Disadvantages:** Inefficient search (O(n)) and sorting (O(n log n)).

**Hash Table**

* **Advantages:** Efficient for insertion and search (O(1)).
* **Disadvantages:** Does not maintain order, requiring additional sorting (O(n log n)).

**Binary Search Tree**

* **Advantages:** Maintains sorted order; efficient search and insertion (O(log n)).
* **Disadvantages:** Can become unbalanced, affecting performance.

## **4. Recommendation**

**Recommended Data Structure:** **Binary Search Tree**

**Justification:**

* Efficient insertion and search operations (O(log n)).
* Maintains sorted order, simplifying the task of printing courses alphabetically.
* Balanced trees provide consistent performance, making them a reliable choice.

This recommendation balances efficiency for search and insertion with the need to maintain a sorted order for course listings.