Southern New Hampshire University

CS-300: Analysis and Design

Maxwell J. Sciola

Professor Gruner

April 7, 2025

1. Pseudocode

**Vector Data Structure Pseudocode**

**TASK: Reading File**

FUNCTION ReadFile(filename)

CREATE fstream fileStream

OPEN fileStream with filename

IF fileStream.open() returns -1 THEN

PRINT "Error: File not found"

RETURN ERROR

ELSE

PRINT "File found"

WHILE not EOF(fileStream) DO

READ line from fileStream

SPLIT line into values

IF number of values < 2 THEN

PRINT "Error: Invalid line format"

RETURN ERROR

ELSE

STORE first two values as parameters

IF number of values >= 3 THEN

FOR each additional parameter (3rd and beyond) DO

IF additional parameter exists in first parameters elsewhere THEN

CONTINUE

ELSE

PRINT "Error: Invalid prerequisite"

RETURN ERROR

END IF

END FOR

END IF

END IF

END WHILE

CLOSE fileStream

END IF

END FUNCTION

**TASK: Create Course Objects**

FUNCTION CreateCourseObjects()

INITIALIZE Vector courseInfo as vector<int>

WHILE not EOF DO

FOR each line in file DO

FOR first and second value in line DO

ADD value to courseInfo (push\_back)

END FOR

IF third value exists THEN

WHILE not new line DO

ADD value to courseInfo (push\_back)

END WHILE

END IF

END FOR

END WHILE

RETURN courseInfo

END FUNCTION

**TASK: Search and Print from Vector**

FUNCTION SearchCourses(courseVector)

PROMPT user for input

ASSIGN input to userInput

FOR each course in courseVector DO

IF userInput equals courseNumber THEN

PRINT course information

FOR each prerequisite in course DO

PRINT prerequisite course information

END FOR

END IF

END FOR

END FUNCTION

**Hash Table Data Structure Pseudocode**

**TASK: Reading File**

FUNCTION ReadFile(filename)

CREATE fstream fileStream

OPEN fileStream with filename

IF fileStream.open() returns -1 THEN

PRINT "Error: File not found"

RETURN ERROR

ELSE

PRINT "File found"

WHILE not EOF(fileStream) DO

READ line from fileStream

SPLIT line into values

IF number of values < 2 THEN

PRINT "Error: Invalid line format"

RETURN ERROR

ELSE

STORE first two values as parameters

IF number of values >= 3 THEN

FOR each additional parameter (3rd and beyond) DO

IF additional parameter exists in first parameters elsewhere THEN

CONTINUE

ELSE

PRINT "Error: Invalid prerequisite"

RETURN ERROR

END IF

END FOR

END IF

END IF

END WHILE

CLOSE fileStream

END IF

END FUNCTION

**TASK: Create Course Objects HashTable**

FUNCTION InitializeHashTable()

INITIALIZE Vector nodes as vector<Node>

CREATE HashTable class

CREATE Insert method for HashTable

END FUNCTION

TASK: Insert Data into HashTable

FUNCTION InsertCourses(filename)

CALL ReadFile(filename)

WHILE not EOF DO

FOR each line in file DO

EXTRACT first and second value

CREATE temp item to hold values

IF third value exists THEN

ADD prerequisite to the course

END IF

CALL Insert method to add course to HashTable

END FOR

END WHILE

END FUNCTION

**TASK: Search and Print from HashTable**

FUNCTION SearchCourse()

PROMPT user for input

ASSIGN input to key

IF key is found in HashTable THEN

PRINT course information

FOR each prerequisite of the course DO

PRINT prerequisite course information

END FOR

ELSE

PRINT "Course not found"

END IF

END FUNCTION

**Binary Search Tree (BST) Data Structure Pseudocode**

**TASK: Reading File**

FUNCTION ReadFile(filename)

CREATE fstream fileStream

OPEN fileStream with filename

IF fileStream.open() returns -1 THEN

PRINT "Error: File not found"

RETURN ERROR

ELSE

PRINT "File found"

WHILE not EOF(fileStream) DO

READ line from fileStream

SPLIT line into courseNumber, courseTitle, prerequisites[]

IF number of values < 2 THEN

PRINT "Error: Invalid file format on line " + LINE\_NUMBER

CONTINUE

ELSE

FOR each prerequisite in prerequisites[] DO

IF prerequisite not in course list THEN

PRINT "Error: Prerequisite " + prerequisite + " not found in course list"

CONTINUE

END IF

END FOR

CREATE new Course object with courseNumber, courseTitle, prerequisites

CALL Insert method to add Course object to BinarySearchTree

END IF

END WHILE

CLOSE fileStream

END IF

END FUNCTION

**TASK: Create Binary Search Tree Structure and Insert Function**

CLASS Course

ATTRIBUTES:

STRING courseNumber

STRING courseTitle

LIST prerequisites

END CLASS

CLASS TreeNode

ATTRIBUTES:

Course data

TreeNode leftChild

TreeNode rightChild

END CLASS

CLASS BinarySearchTree

ATTRIBUTES:

TreeNode root

METHODS:

FUNCTION Insert(courseData)

IF root IS NULL THEN

ASSIGN root = new TreeNode(courseData)

ELSE

CALL InsertRecursive(root, courseData)

END IF

END FUNCTION

FUNCTION InsertRecursive(node, courseData)

IF courseData.courseNumber < node.data.courseNumber THEN

IF node.leftChild IS NULL THEN

ASSIGN node.leftChild = new TreeNode(courseData)

ELSE

CALL InsertRecursive(node.leftChild, courseData)

END IF

ELSE

IF node.rightChild IS NULL THEN

ASSIGN node.rightChild = new TreeNode(courseData)

ELSE

CALL InsertRecursive(node.rightChild, courseData)

END IF

END IF

END FUNCTION

END CLASS

**TASK: Print Course Information in Sorted Order**

FUNCTION PrintInOrder(node)

IF node IS NOT NULL THEN

CALL PrintInOrder(node.leftChild)

PRINT node.data.courseNumber + " " + node.data.courseTitle

PRINT "Prerequisites: " + node.data.prerequisites

CALL PrintInOrder(node.rightChild)

END IF

END FUNCTION

**TASK: Main Execution Flow**

FUNCTION Main()

DECLARE BinarySearchTree courseTree

CALL ReadFile("course\_data.txt")

CALL PrintInOrder(courseTree.root)

END FUNCTION

**Menu System Pseudocode**

**TASK: Display and Handle Menu**

FUNCTION DisplayMenu(dataStructure)

DECLARE bool running = true

WHILE running DO

PRINT "Menu Options:"

PRINT "1. Load file data into the data structure"

PRINT "2. Print ordered list of all Computer Science courses"

PRINT "3. Print course title and prerequisites for a specific course"

PRINT "9. Exit the program"

PROMPT user for input

ASSIGN input to choice

IF choice = 1 THEN

CALL LoadFileData(dataStructure)

ELSE IF choice = 2 THEN

CALL PrintOrderedCourses(dataStructure)

ELSE IF choice = 3 THEN

CALL PrintCourseDetails(dataStructure)

ELSE IF choice = 9 THEN

PRINT "Exiting program"

ASSIGN running = false

ELSE

PRINT "Error: Invalid menu option. Please enter 1, 2, 3, or 9."

END IF

END WHILE

END FUNCTION

**Alphanumeric Vector Data Structure Pseudocode**

**TASK: Sort and Print Courses in Alphanumeric Order**

FUNCTION PrintOrderedCourses(courseVector)

IF courseVector is empty THEN

PRINT "Error: No data loaded. Please load file data first."

ELSE

PRINT "Alphanumerically ordered list of Computer Science courses:"

// Sort the vector by courseNumber (alphanumeric, lowest to highest)

SORT courseVector by courseNumber using comparison (e.g., < operator)

FOR each course in courseVector DO

PRINT course.courseNumber + " " + course.courseTitle

IF course.prerequisites is not empty THEN

PRINT "Prerequisites: " + course.prerequisites

END IF

END FOR

END IF

END FUNCTION

**Alphanumeric Hash Table Data Structure Pseudocode**

**TASK: Sort and Print Courses in** **Alphanumeric Order**

FUNCTION PrintOrderedCourses(hashTable)

IF hashTable is empty THEN

PRINT "Error: No data loaded. Please load file data first."

ELSE

PRINT "Alphanumerically ordered list of Computer Science courses:"

// Extract all courses into a temporary vector for sorting

INITIALIZE Vector tempVector

FOR each entry in hashTable DO

ADD entry.course to tempVector

END FOR

// Sort the vector by courseNumber (alphanumeric, lowest to highest)

SORT tempVector by courseNumber using comparison (e.g., < operator)

FOR each course in tempVector DO

PRINT course.courseNumber + " " + course.courseTitle

IF course.prerequisites is not empty THEN

PRINT "Prerequisites: " + course.prerequisites

END IF

END FOR

END IF

END FUNCTION

**Alphanumeric Binary Search Tree (BST) Data Structure Pseudocode**

**TASK: Print Courses in Alphanumeric Order**

FUNCTION PrintOrderedCourses(bst)

IF bst.root IS NULL THEN

PRINT "Error: No data loaded. Please load file data first."

ELSE

PRINT "Alphanumerically ordered list of Computer Science courses:"

CALL PrintInOrder(bst.root)

END IF

END FUNCTION

FUNCTION PrintInOrder(node)

IF node IS NOT NULL THEN

CALL PrintInOrder(node.leftChild)

PRINT node.data.courseNumber + " " + node.data.courseTitle

IF node.data.prerequisites is not empty THEN

PRINT "Prerequisites: " + node.data.prerequisites

END IF

CALL PrintInOrder(node.rightChild)

END IF

END FUNCTION

1. Evaluation

**Vector Data Structure**

*Reading File Analysis:*

|  |  |  |  |
| --- | --- | --- | --- |
| Code | Line Cost | # Times Executes | Total Cost |
| OPEN fileStream with filename | 1 | 1 | 1 |
| IF fileStream.open() returns -1 | 1 | 1 | 1 |
| WHILE not EOF(fileStream) DO | 1 | n+1 | n+1 |
| READ line from fileStream | 1 | n | n |
| SPLIT line into values | 1 | n | n |
| IF number of values < 2 | 1 | n | n |
| STORE first two values as parameters | 1 | n | n |
| IF number of values >= 3 | 1 | n | n |
| FOR each additional parameter | 1 | n\*p | n\*p |
| IF additional parameter exists | 1 | n\*p | n\*p |
| CLOSE fileStream | 1 | 1 | 1 |
| Total Cost |  |  | 5n+2n\*p+3 |
| **Runtime** |  |  | O(n\*p) |

*Creating Course Objects Analysis:*

|  |  |  |  |
| --- | --- | --- | --- |
| Code | Line Cost | # Times Executes | Total Cost |
| INITIALIZE Vector courseInfo | 1 | 1 | 1 |
| WHILE not EOF DO | 1 | n+1 | n+1 |
| FOR each line in file DO | 1 | n | n |
| FOR first and second value | 1 | 2n | 2n |
| ADD value to courseInfo | 1 | 2n | 2n |
| IF third value exists | 1 | n | n |
| WHILE not new line DO | 1 | n\*p | n\*p |
| ADD value to courseInfo | 1 | n\*p | n\*p |
| Total Cost |  |  | 6n+2n\*p+1 |
| **Runtime** |  |  | O(n\*p) |

**Hash Table Data Structure**

*Analysis:*

|  |  |  |  |
| --- | --- | --- | --- |
| Code | Line Cost | # Times Executes | Total Cost |
| CALL ReadFile(filename) | O(n\*p) | 1 | O(n\*p) |
| WHILE not EOF DO | 1 | n+1 | n+1 |
| FOR each line in file DO | 1 | n | n |
| EXTRACT first and second value | 1 | n | n |
| CREATE temp item | 1 | n | n |
| IF third value exists | 1 | n | n |
| ADD prerequisite to course | 1 | n\*p | n\*p |
| CALL Insert method | 1 | n | n |
| Total Cost |  |  | O(n\*p) |

* Runtime is O(n²)

**Binary Search Tree Data Structure**

|  |  |  |  |
| --- | --- | --- | --- |
| Code | Line Cost | # Times Executes | Total Cost |
| WHILE not EOF DO | 1 | n+1 | n+1 |
| READ and SPLIT line | 1 | n | n |
| IF number of values < 2 | 1 | n | n |
| FOR each prerequisite | 1 | n\*p | n\*p |
| IF prerequisite not in course list | 1 | n\*p | n\*p |
| CREATE new Course object | 1 | n | n |
| CALL Insert method | O(log n) to O(n) | n | O(log n) to O(n) |
| Total Cost |  |  | O(n\*p + n log n) to O(n\*p + n²) |

* Worst case Big O is O(n²)

**Advantages and Disadvantages of Each Structure**

**Vector**

* *Advantages*
  + Implementation is simple.
  + Debugging is easy.
  + After loading, sequential access is fast.
* *Disadvantages*
  + Runtime of O(n²).
  + For ordered output, sorting needs to be (O(n log n)).

**Hash Table**

* *Advantages*
  + Case insertion/look up has an average of O(1).
  + O(n) memory usage.
* *Disadvantages*
  + Needs to prerequisite validate causing a O(n²) runtime.
  + Needs extra sorting to produce an alphanumeric output.

**Binary Search Tree**

* *Advantages*
  + Perfect for alphanumeric outputs due to being naturally sorted.
  + Insertion runtime of O(n log n).
  + O(n) memory usage.
* *Disadvantages*
  + Runtime of O(n²).

**Recommendation**

Based on all the information I have gathered, the recommended data structure for this assignment I would pick is a Binary Search Tree (BST). Even though all three structures have their advantages, BST offers the most potential for optimization. The total runtime can be reduced because there is room for improvement in the insertion runtime, unlike for Vectors or Hash Tables, which will remain O(n²) no matter what. A key advantage of BSTs is their inherent ability to maintain alphanumeric order, making it perfect for the requirement to print courses in sorted order. Also, Vectors and Hash Tables need O(n log n) for sorting, while BST traversal occurs in O(n) time. All three use O(n) space for memory usage, so memory isn’t an issue for which one I pick. However, BSTs have a small recursion overhead of O(log n), but this is so small that it does not even make a difference. Also, BSTs have high levels of potential optimization when you combine a hash set for prerequisite validation. This change would cause an O(n) space cost but allow for O(1) lookups, causing the BSTs to have the fastest option of O(n log n) for both sorting and loading. After considering everything, I believe BSTs are a perfect fit for ABCU’s advising program because they provide natural ordering and the ability to improve.