Assignment Six

Keyneisha Davion Mcnealey

Professor Othman

April 13, 2025

Course structure/object is defined as:

```
STRUCTURE Course
   TEXT courseNumber
   TEXT courseTitle
   ARRAY of TEXT prerequisites
END STRUCTURE
```

Pseudocode

A. Resubmitted and Updated Pseudocode

1. Vector

- **Pseudocode for Opening, Reading, Parsing, and Validating the File**

```
FUNCTION loadCourseDataIntoVector(fileName AS TEXT) RETURNS VECTOR of Course
   DECLARE courseVector AS VECTOR of Course

   OPEN fileName FOR READING
   IF FILE_OPEN_ERROR THEN
      DISPLAY "Error: Could not open file " + fileName
      RETURN courseVector
   ENDIF

   WHILE NOT END_OF_FILE(fileName) DO
      DECLARE line AS TEXT
      READLINE fileName INTO line

      DECLARE parts AS ARRAY of TEXT
      parts = SPLIT(line, ",")

      IF LENGTH(parts) < 2 THEN
         DISPLAY "Error: Invalid line format: " + line
         CONTINUE
      ENDIF

      DECLARE courseNumber AS TEXT
      DECLARE courseTitle AS TEXT
```

```
        courseNumber = TRIM(parts[0])
        courseTitle = TRIM(parts[1])

        DECLARE currentCourse AS Course
        currentCourse.courseNumber = courseNumber
        currentCourse.courseTitle = courseTitle

        DECLARE prerequisites AS ARRAY of TEXT
        prerequisites = CREATE_EMPTY_ARRAY

        FOR i = 2 TO LENGTH(parts) - 1 DO
            DECLARE prereqNumber AS TEXT
            prereqNumber = TRIM(parts[i])

            // Validation: Check if prerequisite exists (inefficient in vector)
            DECLARE found AS BOOLEAN
            found = FALSE
            FOR EACH course IN courseVector DO
                IF course.courseNumber == prereqNumber THEN
                    found = TRUE
                    BREAK
                ENDIF
            ENDFOR
            IF NOT found THEN
                DISPLAY "Error: Prerequisite " + prereqNumber + " does not exist for
course " + courseNumber
                CONTINUE
            ENDIF

            APPEND prereqNumber TO prerequisites
        ENDFOR
        currentCourse.prerequisites = prerequisites

        APPEND currentCourse TO courseVector
    ENDWHILE

    CLOSE fileName
    RETURN courseVector
END FUNCTION
```

- **Pseudocode to Show How to Create Course Objects and Store Them**

```
//(Inside loadCourseDataIntoVector)
DECLARE currentCourse AS Course
currentCourse.courseNumber = courseNumber
currentCourse.courseTitle = courseTitle
currentCourse.prerequisites = prerequisites
APPEND currentCourse TO courseVector
```

- **Pseudocode to Print Out Course Information and Prerequisites**

```
FUNCTION printCourseInfo(courses AS VECTOR of Course, courseNumberToFind
AS TEXT)
   DECLARE found AS BOOLEAN
   found = FALSE

   FOR EACH course IN courses DO
      IF course.courseNumber == courseNumberToFind THEN
         found = TRUE
         DISPLAY "Course Number: " + course.courseNumber
         DISPLAY "Course Title: " + course.courseTitle
         IF LENGTH(course.prerequisites) > 0 THEN
            DISPLAY "Prerequisites:"
            FOR EACH prereq IN course.prerequisites DO
               DISPLAY "  - " + prereq
            ENDFOR
         ELSE
            DISPLAY "  No prerequisites."
         ENDIF
         BREAK
      ENDIF
   ENDFOR

   IF NOT found THEN
      DISPLAY "Course " + courseNumberToFind + " not found."
   ENDIF
END FUNCTION
```

2. Hash Table

- **Pseudocode for Opening, Reading, Parsing, and Validating the File**

```
FUNCTION loadCourseDataIntoHashTable(fileName AS TEXT) RETURNS
HASH_TABLE<TEXT, Course>
  DECLARE courseHashTable AS HASH_TABLE<TEXT, Course>

  OPEN fileName FOR READING
  IF FILE_OPEN_ERROR THEN
    DISPLAY "Error: Could not open file " + fileName
    RETURN courseHashTable
  ENDIF

  WHILE NOT END_OF_FILE(fileName) DO
    DECLARE line AS TEXT
    READLINE fileName INTO line

    DECLARE parts AS ARRAY of TEXT
    parts = SPLIT(line, ",")

    IF LENGTH(parts) < 2 THEN
      DISPLAY "Error: Invalid line format: " + line
      CONTINUE
    ENDIF

    DECLARE courseNumber AS TEXT
    DECLARE courseTitle AS TEXT
    courseNumber = TRIM(parts[0])
    courseTitle = TRIM(parts[1])

    DECLARE currentCourse AS Course
    currentCourse.courseNumber = courseNumber
    currentCourse.courseTitle = courseTitle

    DECLARE prerequisites AS ARRAY of TEXT
    prerequisites = CREATE_EMPTY_ARRAY

    FOR i = 2 TO LENGTH(parts) - 1 DO
```

```
        DECLARE prereqNumber AS TEXT
        prereqNumber = TRIM(parts[i])

        // Validation: Check if prerequisite exists
        IF NOT courseHashTable.contains(prereqNumber) THEN
            DISPLAY "Error: Prerequisite " + prereqNumber + " does not exist for
course " + courseNumber
            CONTINUE
        ENDIF

        APPEND prereqNumber TO prerequisites
      ENDFOR
      currentCourse.prerequisites = prerequisites

      courseHashTable.insert(courseNumber, currentCourse)
   ENDWHILE

   CLOSE fileName
   RETURN courseHashTable
END FUNCTION
```

- **Pseudocode to Show How to Create Course Objects and Store Them**

```
//(Inside loadCourseDataIntoHashTable)
DECLARE currentCourse AS Course
currentCourse.courseNumber = courseNumber
currentCourse.courseTitle = courseTitle
currentCourse.prerequisites = prerequisites
courseHashTable.insert(courseNumber, currentCourse)
```

- **Pseudocode to Print Out Course Information and Prerequisites**

```
FUNCTION printCourseInfo(courses AS HASH_TABLE<TEXT, Course>,
courseNumberToFind AS TEXT)
   IF courses.contains(courseNumberToFind) THEN
      DECLARE foundCourse AS Course
      foundCourse = courses.get(courseNumberToFind)

      DISPLAY "Course Number: " + foundCourse.courseNumber
```

```
    DISPLAY "Course Title: " + foundCourse.courseTitle

    IF LENGTH(foundCourse.prerequisites) > 0 THEN
        DISPLAY "Prerequisites:"
        FOR EACH prereq IN foundCourse.prerequisites DO
            DISPLAY "  - " + prereq
        ENDFOR
    ELSE
        DISPLAY "  No prerequisites."
    ENDIF
  ELSE
    DISPLAY "Course " + courseNumberToFind + " not found."
  ENDIF
END FUNCTION
```

3. Binary Search Tree

- **Pseudocode for Opening, Reading, Parsing, and Validating the File**

```
FUNCTION loadCourseDataIntoTree(fileName AS TEXT) RETURNS
BinarySearchTree
  DECLARE courseTree AS BinarySearchTree

  OPEN fileName FOR READING
  IF FILE_OPEN_ERROR THEN
    DISPLAY "Error: Could not open file " + fileName
    RETURN courseTree
  ENDIF

  WHILE NOT END_OF_FILE(fileName) DO
    DECLARE line AS TEXT
    READLINE fileName INTO line

    DECLARE parts AS ARRAY of TEXT
    parts = SPLIT(line, ",")

    IF LENGTH(parts) < 2 THEN
        DISPLAY "Error: Invalid line format: " + line
        CONTINUE
```

```
        ENDIF

        DECLARE courseNumber AS TEXT
        DECLARE courseTitle AS TEXT
        courseNumber = TRIM(parts[0])
        courseTitle = TRIM(parts[1])

        DECLARE currentCourse AS Course
        currentCourse.courseNumber = courseNumber
        currentCourse.courseTitle = courseTitle

        DECLARE prerequisites AS ARRAY of TEXT
        prerequisites = CREATE_EMPTY_ARRAY

        FOR i = 2 TO LENGTH(parts) - 1 DO
            DECLARE prereqNumber AS TEXT
            prereqNumber = TRIM(parts[i])

            // Validation: Check if prerequisite exists
            IF NOT courseTree.contains(prereqNumber) THEN
                DISPLAY "Error: Prerequisite " + prereqNumber + " does not exist for
course " + courseNumber
                CONTINUE
            ENDIF

            APPEND prereqNumber TO prerequisites
        ENDFOR
        currentCourse.prerequisites = prerequisites

        courseTree.insert(currentCourse)
    ENDWHILE

    CLOSE fileName
    RETURN courseTree
END FUNCTION
```

- **Pseudocode to Show How to Create Course Objects and Store Them**

```
//(Inside loadCourseDataIntoTree)
```

```
DECLARE currentCourse AS Course
currentCourse.courseNumber = courseNumber
currentCourse.courseTitle = courseTitle
currentCourse.prerequisites = prerequisites
courseTree.insert(currentCourse)
```

- **Pseudocode to Print Out Course Information and Prerequisites**

```
FUNCTION printCourseInfo(courseTree AS BinarySearchTree,
courseNumberToFind AS TEXT)
    DECLARE foundCourse AS Course
    foundCourse = courseTree.search(courseNumberToFind)

    IF foundCourse.courseNumber is empty THEN  // Check if courseNumber is
empty (or some other indicator of not found)
        DISPLAY "Course " + courseNumberToFind + " not found."
        RETURN
    ENDIF

    DISPLAY "Course Number: " + foundCourse.courseNumber
    DISPLAY "Course Title: " + foundCourse.courseTitle

    IF LENGTH(foundCourse.prerequisites) > 0 THEN
        DISPLAY "Prerequisites:"
        FOR EACH prereq IN foundCourse.prerequisites DO
            DISPLAY "  - " + prereq
        ENDFOR
    ELSE
        DISPLAY "  No prerequisites."
    ENDIF
END FUNCTION
```

B. Pseudocode for Menu

```
FUNCTION main()

    DECLARE courses AS DataStructure // Change DataStructure based on implementation (Vector, HashTable, BinarySearchTree)
    DECLARE fileName AS TEXT
    DECLARE userChoice AS INTEGER
    DECLARE searchCourseNumber AS TEXT

    fileName = "courses.txt" // Or get from user input

    WHILE userChoice != 9 DO
        DISPLAY "Menu:"
        DISPLAY "1. Load Course Data"
        DISPLAY "2. Print Course List (Alphanumeric)"
        DISPLAY "3. Print Course Information"
        DISPLAY "9. Exit"
        PROMPT "Enter choice: "
        READ userChoice

        SWITCH userChoice
            CASE 1:
                courses = loadCourseData(fileName) // Call the appropriate load function
                BREAK
            CASE 2:
                printSortedCourseList(courses) // Call the appropriate print function
                BREAK
            CASE 3:
                PROMPT "Enter course number: "
                READ searchCourseNumber
                printCourseInfo(courses, searchCourseNumber)
                BREAK
            CASE 9:
                DISPLAY "Exiting..."
                BREAK
            DEFAULT:
                DISPLAY "Invalid choice. Please try again."
```

```
        ENDSWITCH
    ENDWHILE

END FUNCTION
```

C. Pseudocode for Printing Sorted Course List

1. Vector

```
FUNCTION printSortedCourseList(courses AS VECTOR of Course)
    // 1. Sort the Vector
    SORT courses BY courseNumber  // Assuming a built-in sort function or you implement one

    // 2. Iterate and Print
    FOR EACH course IN courses DO
        DISPLAY course.courseNumber + " - " + course.courseTitle
    ENDFOR
END FUNCTION
```

2. Hash Table

```
FUNCTION printSortedCourseList(courses AS HASH_TABLE<TEXT, Course>)
    // 1. Get All Course Numbers (Keys)
    DECLARE courseNumbers AS ARRAY of TEXT
    courseNumbers = courses.getKeys()  // Assuming a getKeys() method exists

    // 2. Sort the Course Numbers
    SORT courseNumbers  // Sort the array of course numbers

    // 3. Iterate and Print
    FOR EACH courseNumber IN courseNumbers DO
        DECLARE course AS Course
        course = courses.get(courseNumber)
        DISPLAY course.courseNumber + " - " + course.courseTitle
    ENDFOR
END FUNCTION
```

3. Binary Search Tree

```
// Helper function for in-order traversal (BST already sorted)
FUNCTION inOrderTraversal(node AS TreeNode)
    IF node is not NULL THEN
        inOrderTraversal(node.leftChild)
        DISPLAY node.data.courseNumber + " - " + node.data.courseTitle
        inOrderTraversal(node.rightChild)
    ENDIF
END FUNCTION

FUNCTION printSortedCourseList(courses AS BinarySearchTree)
    // 1. Perform In-Order Traversal (BST is inherently sorted)
    inOrderTraversal(courses.root)  // Assuming 'root' is the root node
END FUNCTION
```

Evaluation and Runtime Analysis

A. Runtime Analysis: Reading File and Creating Course Objects

Assume:

- n courses in the file.
- m is the maximum number of prerequisites for any single course.

Operation	Vector Cost/Line	Vector Execution Count	Hash Table Cost/Line	Hash Table Execution Count	Tree Cost/Line	Tree Execution Count
OPEN fileName FOR READING	1	1	1	1	1	1
WHILE NOT END_OF_FILE(fileName)	1	n + 1	1	n + 1	1	n + 1
READLINE fileName INTO line	1	n	1	n	1	n
parts = SPLIT(line, ",")	1	n	1	n	1	n
IF LENGTH(parts) < 2	1	n	1	n	1	n
courseNumber = TRIM(parts[0])	1	n	1	n	1	n
courseTitle =	1	n	1	n	1	n

TRIM(parts[1])						
DECLARE currentCourse AS Course	1	n	1	n	1	n
currentCourse.courseNumber = courseNumber	1	n	1	n	1	n
currentCourse.courseTitle = courseTitle	1	n	1	n	1	n
prerequisites = CREATE_EMPTY_ARRAY	1	n	1	n	1	n
FOR i = 2 TO LENGTH(parts) - 1	1	n * m (worst case)	1	n * m (worst case)	1	n * m (worst case)
IF NOT courseTree.contains(prereqNumber)	n	n * m (worst case)	1	n * m (worst case)	log n	n * m (worst case)
APPEND prereqNumber TO prerequisites	1	n * m (worst case)	1	n * m (worst case)	1	n * m (worst case)

currentCourse.prerequisites = prerequisites	1	n	1	n	1	n
courseVector.append(currentCourse)	1	n	1	n	log n	n
CLOSE fileName	1	1	1	1	1	1
Total Cost		**O(n^2 * m)**		**O(n * m)**		**O(n * m * log n)**

Worst-Case Big O Analysis:

- **Vector:** O(n² * m) - The nested loop for prerequisite validation in the vector is the dominant factor.
- **Hash Table:** O(n * m) - Hash table lookups (contains) are typically O(1) on average, making the prerequisite validation efficient.
- **Binary Search Tree:** O(n * m * log n) - BST lookups (contains) are O(log n), so the nested loop becomes O(n * m * log n).

B. Data Structure Analysis

1. Vector

- **Advantages:**
  - Simple to implement.
  - Efficient for sequential access.
- **Disadvantages:**
  - Inefficient for searching (O(n) on average, O(n) worst-case) and therefore prerequisite validation.
  - Inefficient for insertion and deletion if order needs to be maintained.

2. Hash Table

- **Advantages:**
  - Very efficient for searching (O(1) on average, O(n) worst-case, but unlikely).

    - Fast insertion and deletion (O(1) on average).
- **Disadvantages:**
    - Can have collisions, leading to worst-case O(n) search time, but good hash functions minimize this.
    - Sorting is not inherent and requires extra steps.

3. Binary Search Tree

- **Advantages:**
    - Efficient for searching, insertion, and deletion (O(log n) on average).
    - Inherent ordering of elements, making sorted output easier.
- **Disadvantages:**
    - Can become unbalanced, leading to worst-case O(n) performance.

C. Recommendation

Based on the runtime analysis and data structure analysis, the **Hash Table** is the most suitable data structure for this application.

Justification:

- The primary operation that dominates the runtime is checking for prerequisite existence. The hash table provides the most efficient average-case performance for this operation (O(1)), compared to O(n) for the vector and O(log n) for the BST.
- While the BST offers sorted output, the application requires an *alphanumerically ordered list*, which can be achieved by sorting the keys of the hash table separately (which is still more efficient than repeated searching).
- The vector's inefficiency in searching makes it the least desirable option.