ABCU Course Information System

Complete Pseudocode and Runtime Analysis

This is a complete Unified Document of All Milestones and Project One

Student Name: Zion D. Kinniebrew- Jenkin s
Course: CS-300
Date: 10/10/2025

PART 1: PSEUDOCODE IMPLEMENTATIONS

A. VECTOR IMPLEMENTATION

Course Structure:

```
STRUCTURE Course
   STRING courseNumber
   STRING title
   VECTOR<STRING> prerequisites
END STRUCTURE
```

Load Courses Function:

```
PROCEDURE LoadCourses(STRING filePath)
   DEFINE courseList AS VECTOR<Course>
   DEFINE allCourseNumbers AS VECTOR<STRING>

   OPEN file at filePath
   IF file cannot be opened THEN
      PRINT "Error: File could not be opened."
      RETURN NULL
   END IF

   WHILE NOT end of file
      STRING currentLine = READ line from file
      VECTOR<STRING> tokens = SPLIT currentLine by ','

      IF size of tokens < 2 THEN
         PRINT "Warning: Skipping malformed line"
```

```
        CONTINUE
      END IF

      Course newCourse
      newCourse.courseNumber = tokens[0]
      newCourse.title = tokens[1]

      FOR i FROM 2 TO size of tokens - 1
        ADD tokens[i] TO newCourse.prerequisites
      END FOR

      ADD newCourse TO courseList
      ADD newCourse.courseNumber TO allCourseNumbers
    END WHILE
    CLOSE file

    FOR EACH course IN courseList
      FOR EACH prereq IN course.prerequisites
        IF prereq NOT IN allCourseNumbers THEN
          PRINT "Error: Invalid prerequisite"
        END IF
      END FOR
    END FOR

    RETURN courseList
END PROCEDURE
```

Print All Courses (Sorted):

```
PROCEDURE PrintAllCourses(VECTOR<Course> courseList)
    SORT courseList BY courseNumber
    FOR EACH course IN courseList
      PRINT course.courseNumber + ", " + course.title
    END FOR
END PROCEDURE
```

Print Single Course:

```
PROCEDURE PrintCourseInfo(STRING courseId, VECTOR<Course> courseList)
    FOR EACH course IN courseList
      IF course.courseNumber EQUALS courseId THEN
        PRINT course.courseNumber + ", " + course.title
        PRINT "Prerequisites: "
```

```
        IF course.prerequisites IS EMPTY THEN
            PRINT "None"
        ELSE
            FOR EACH prereq IN course.prerequisites
                PRINT prereq
            END FOR
        END IF
        BREAK
    END IF
  END FOR
END PROCEDURE
```

Menu System:

```
PROCEDURE Main()
  VECTOR<Course> allCourses
  BOOLEAN dataLoaded = false

  LOOP
    PRINT "1. Load Data Structure"
    PRINT "2. Print Course List"
    PRINT "3. Print Course"
    PRINT "9. Exit"
    GET userChoice

    SWITCH userChoice
      CASE 1:
        allCourses = LoadCourses("courses.csv")
        dataLoaded = true
      CASE 2:
        IF dataLoaded THEN PrintAllCourses(allCourses)
      CASE 3:
        IF dataLoaded THEN
          GET courseId
          PrintCourseInfo(courseId, allCourses)
        END IF
      CASE 9:
        EXIT LOOP
    END SWITCH
  END LOOP
END PROCEDURE
```

B. HASH TABLE IMPLEMENTATION

Course Structure:

```
STRUCTURE Course
   STRING courseNumber
   STRING courseTitle
   VECTOR<STRING> prerequisites
END STRUCTURE

DEFINE hashTable AS HASH_TABLE<Course> (size = 127)
```

Load Courses Function:

```
FUNCTION LoadCoursesFromFile(STRING fileName)
   DEFINE allCourseNumbers AS SET<STRING>

   file = OPEN fileName
   lines = READ all lines from file
   CLOSE file

   FOR EACH line IN lines
      tokens = SPLIT line by ','
      IF tokens.size >= 2 THEN
         ADD tokens[0] TO allCourseNumbers
      END IF
   END FOR

   FOR EACH line IN lines
      tokens = SPLIT line by ','
      IF tokens.size < 2 THEN CONTINUE

      course = CREATE Course
      course.courseNumber = tokens[0]
      course.courseTitle = tokens[1]

      FOR i = 2 TO tokens.size - 1
         IF tokens[i] IN allCourseNumbers THEN
            ADD tokens[i] TO course.prerequisites
         END IF
      END FOR

      INSERT course INTO hashTable
```

```
    END FOR

    RETURN true
END FUNCTION
```

Hash and Search Functions:

```
FUNCTION CalculateHash(STRING key)
    hash = 0
    FOR EACH character IN key
        hash = (hash * 31 + ASCII(character)) MOD 127
    END FOR
    RETURN hash
END FUNCTION

FUNCTION SearchCourse(STRING courseNumber)
    index = CalculateHash(courseNumber)
    FOR EACH course IN hashTable[index]
        IF course.courseNumber EQUALS courseNumber THEN
            RETURN course
        END IF
    END FOR
    RETURN NULL
END FUNCTION
```

Print Functions:

```
FUNCTION PrintAllCourses()
    allCourses = CREATE empty vector

    FOR i = 0 TO 126
        FOR EACH course IN hashTable[i]
            ADD course TO allCourses
        END FOR
    END FOR

    SORT allCourses BY courseNumber

    FOR EACH course IN allCourses
        PRINT course.courseNumber + ", " + course.courseTitle
    END FOR
END FUNCTION
```

```
FUNCTION PrintCourseInformation(STRING courseNumber)
   course = SearchCourse(courseNumber)
   IF course is NULL THEN
      PRINT "Course not found"
   ELSE
      PRINT course.courseNumber + ", " + course.courseTitle
      PRINT "Prerequisites: "
      IF course.prerequisites is empty THEN
         PRINT "None"
      ELSE
         FOR EACH prereq IN course.prerequisites
            PRINT prereq
         END FOR
      END IF
   END IF
END FUNCTION
```

Menu System:

```
FUNCTION Main()
   dataLoaded = false

   WHILE true
      PRINT "1. Load Data Structure"
      PRINT "2. Print Course List"
      PRINT "3. Print Course"
      PRINT "9. Exit"
      INPUT choice

      SWITCH choice
         CASE 1:
            dataLoaded = LoadCoursesFromFile("courses.csv")
         CASE 2:
            IF dataLoaded THEN PrintAllCourses()
         CASE 3:
            IF dataLoaded THEN
               INPUT courseNumber
               PrintCourseInformation(courseNumber)
            END IF
         CASE 9:
            EXIT
      END SWITCH
   END WHILE
END FUNCTION
```

C. BINARY SEARCH TREE IMPLEMENTATION

Data Structures:

```
STRUCTURE Course
   STRING courseNumber
   STRING courseTitle
   VECTOR<STRING> prerequisites
END STRUCTURE

STRUCTURE Node
   Course course
   Node* left
   Node* right
END STRUCTURE

CLASS BinarySearchTree
   Node* root

   METHOD Insert(Course course)
   METHOD InOrderTraversal()
   METHOD Search(STRING courseNumber)
END CLASS
```

Load Courses Function:

```
FUNCTION LoadCoursesFromFile(STRING filename, BinarySearchTree& bst)
   file = OPEN filename
   IF NOT file.isOpen THEN RETURN false

   tempCourses = CREATE vector<Course>
   allCourseNumbers = CREATE vector<STRING>

   WHILE NOT end of file
      line = READ line
      tokens = SPLIT line by ','

      IF tokens.size < 2 THEN
         PRINT "Error: Missing data"
         RETURN false
      END IF
```

```
        course = CREATE Course
        course.courseNumber = tokens[0]
        course.courseTitle = tokens[1]

        FOR i = 2 TO tokens.size - 1
            ADD tokens[i] TO course.prerequisites
        END FOR

        ADD course TO tempCourses
        ADD course.courseNumber TO allCourseNumbers
    END WHILE
    CLOSE file

    FOR EACH course IN tempCourses
        FOR EACH prereq IN course.prerequisites
            IF prereq NOT IN allCourseNumbers THEN
                PRINT "Error: Invalid prerequisite"
                RETURN false
            END IF
        END FOR
    END FOR

    FOR EACH course IN tempCourses
        bst.Insert(course)
    END FOR

    RETURN true
END FUNCTION
```

BST Operations:

```
METHOD Insert(Course course)
    IF root is nullptr THEN
        root = NEW Node(course)
    ELSE
        InsertHelper(root, course)
    END IF
END METHOD

FUNCTION InsertHelper(Node* node, Course course)
    IF course.courseNumber < node->course.courseNumber THEN
        IF node->left is nullptr THEN
            node->left = NEW Node(course)
        ELSE
```

```
            InsertHelper(node->left, course)
        END IF
    ELSE
        IF node->right is nullptr THEN
            node->right = NEW Node(course)
        ELSE
            InsertHelper(node->right, course)
        END IF
    END IF
END FUNCTION

METHOD InOrderTraversal()
    InOrderHelper(root)
END METHOD

FUNCTION InOrderHelper(Node* node)
    IF node is nullptr THEN RETURN

    InOrderHelper(node->left)
    PRINT node->course.courseNumber + ", " + node->course.courseTitle
    InOrderHelper(node->right)
END FUNCTION

METHOD Search(STRING courseNumber)
    result = SearchHelper(root, courseNumber)
    IF result is nullptr THEN
        PRINT "Course not found"
    ELSE
        PRINT result->course.courseNumber + ", " + result->course.courseTitle
        PRINT "Prerequisites: "
        IF result->course.prerequisites.size > 0 THEN
            FOR EACH prereq IN result->course.prerequisites
                PRINT prereq
            END FOR
        ELSE
            PRINT "None"
        END IF
    END IF
END METHOD

FUNCTION SearchHelper(Node* node, STRING courseNumber)
    IF node is nullptr THEN RETURN nullptr
    IF node->course.courseNumber == courseNumber THEN RETURN node
```

```
    IF courseNumber < node->course.courseNumber THEN
        RETURN SearchHelper(node->left, courseNumber)
    ELSE
        RETURN SearchHelper(node->right, courseNumber)
    END IF
END FUNCTION
```

Menu System:

```
FUNCTION Main()
    bst = CREATE BinarySearchTree
    dataLoaded = false

    WHILE true
        PRINT "1. Load Data Structure"
        PRINT "2. Print Course List"
        PRINT "3. Print Course"
        PRINT "9. Exit"
        INPUT choice

        SWITCH choice
            CASE 1:
                dataLoaded = LoadCoursesFromFile("courses.csv", bst)
            CASE 2:
                IF dataLoaded THEN bst.InOrderTraversal()
            CASE 3:
                IF dataLoaded THEN
                    INPUT courseNumber
                    bst.Search(courseNumber)
                END IF
            CASE 9:
                EXIT
        END SWITCH
    END WHILE
END FUNCTION
```

PART 2: RUNTIME ANALYSIS

Big O Analysis Chart

Data Structure	Load & Create Objects	Search Course	Print All (Sorted)	Space

Vector	O(n^2)	O(n)	O(n log n)	O(n)
Hash Table	O(n)	O(1) average	O(n log n)	O(n)
Binary Search Tree	O(n log n)	O(log n) average	O(n)	O(n)

Detailed Analysis

Vector - O(n^2): Pass 1 reads n lines (5n operations). Pass 2 validates prerequisites with nested loops: for each course (n), check each prerequisite (m) against all course numbers (n), resulting in n x m x n operations. Total: O(n^2).

Hash Table - O(n): Pass 1 collects course numbers in O(1) hash set (2n operations). Pass 2 validates prerequisites with O(1) lookups and inserts with O(1) hash operations. Total: O(n) linear time.

BST - O(n log n): Pass 1 reads n lines (3n). Pass 2 validates with linear search (n^2 x m). Pass 3 inserts n courses with O(log n) each (n log n). Dominated by insertion: O(n log n).

PART 3: DATA STRUCTURE EVALUATION

Vector

Advantages: Simple implementation, efficient memory, easy to maintain, good for small datasets. **Disadvantages:** Slow O(n) search, inefficient O(n^2) validation, poor scalability, requires sorting before display.

Hash Table

Advantages: Fast O(1) search, efficient O(n) loading, excellent for frequent lookups, scales well. **Disadvantages:** Requires sorting for display (O(n log n)), memory overhead, collision handling complexity, not naturally ordered.

Binary Search Tree

Advantages: Naturally sorted, efficient O(n) in-order traversal, good O(log n) search, no sorting needed. **Disadvantages:** Can degrade to O(n) if unbalanced, complex implementation, slower O(log n) insertion, pointer overhead.

PART 4: RECOMMENDATION

Recommended: Hash Table

The hash table provides optimal performance for the primary use case: advisors searching individual courses. With O(1) average lookup time and O(n) loading, it outperforms both vector (O(n^2) load, O(n) search) and BST (O(n log n) load, O(log n) search). While it requires O(n log n) sorting for full list display, this operation is infrequent compared to individual searches. The hash table optimizes for the common case, delivering the best user experience for advisors who primarily lookup specific courses multiple times per session.