CS 300 Project One: Pseudocode and Runtime Analysis

Michael Rodman

Southern New Hampshire University

Date: 10/12/2025

Vector Pseudocode

```
STRUCT Course
   courseNumber : string
   name : string
   prerequisites : Vector<string>


FUNCTION loadCourses(filePath : string) → (Vector<Course>, Vector<string> errors)
   errors ← empty Vector<string>
   lines  ← readAllLines(filePath)
   IF lines is empty THEN
      errors.push_back("File is empty or missing: " + filePath)
      RETURN (empty Vector<Course>, errors)
   ENDIF
```

```
rawRecords ← Vector<Vector<string>>
seen       ← Set<string>

lineNum ← 0
FOR each line IN lines
   lineNum ← lineNum + 1
   IF trim(line) = "" THEN CONTINUE

   tokens ← split(line, ',')
   FOR i FROM 0 TO tokens.size()-1
      tokens[i] ← trim(tokens[i])
   ENDFOR

   IF tokens.size() < 2 THEN
      errors.push_back("Line " + toString(lineNum) + ": missing course number or name")
      CONTINUE
   ENDIF

   courseNum  ← tokens[0]
   courseName ← tokens[1]
   IF courseNum = "" OR courseName = "" THEN
      errors.push_back("Line " + toString(lineNum) + ": invalid course format")
      CONTINUE
   ENDIF
```

```
    IF courseNum IN seen THEN
      errors.push_back("Duplicate course number: " + courseNum)
    ELSE
      seen.insert(courseNum)
    ENDIF

    rawRecords.push_back(tokens)
  ENDFOR

  courses ← Vector<Course>
  FOR each rec IN rawRecords
    course ← new Course
    course.courseNumber ← rec[0]
    course.name         ← rec[1]
    FOR i FROM 2 TO rec.size()-1
      prereq ← rec[i]
      IF prereq ≠ "" AND prereq NOT IN seen THEN
        errors.push_back("Prerequisite " + prereq + " not found for course " +
course.courseNumber)
      ENDIF
      IF prereq ≠ "" THEN
        course.prerequisites.push_back(prereq)
      ENDIF
    ENDFOR
    courses.push_back(course)
```

```
  ENDFOR

  RETURN (courses, errors)
END FUNCTION

FUNCTION searchCourse(courses : Vector<Course>, courseNumber : string)
  found ← FALSE
  FOR each c IN courses
    IF c.courseNumber = courseNumber THEN
      found ← TRUE
      printCourse(c)
      IF c.prerequisites.size() = 0 THEN
        PRINT "Prerequisites: None"
      ELSE
        PRINT "Prerequisites:"
        FOR each p IN c.prerequisites
          pc ← findCourseByNumber(courses, p)
          IF pc ≠ NULL THEN
            PRINT "  - " + pc.courseNumber + ": " + pc.name
          ELSE
            PRINT "  - " + p
          ENDIF
        ENDFOR
      ENDIF
      BREAK
```

```
    ENDIF
  ENDFOR
  IF NOT found THEN
    PRINT "Course " + courseNumber + " not found."
  ENDIF
END FUNCTION

FUNCTION findCourseByNumber(courses : Vector<Course>, courseNumber : string) →
Course or NULL
  FOR each c IN courses
    IF c.courseNumber = courseNumber THEN RETURN c
  ENDFOR
  RETURN NULL
END FUNCTION

FUNCTION printCourse(c : Course)
  PRINT c.courseNumber + ", " + c.name
END FUNCTION

FUNCTION printAllCourses(courses : Vector<Course>)
  temp ← copy of courses
  sort(temp by courseNumber ascending)
  FOR each c IN temp
    PRINT c.courseNumber + ", " + c.name
```

```
  ENDFOR
END FUNCTION
```

Hash Table Pseudocode

```
STRUCT Course
  courseNumber : string
  name         : string
  prerequisites: list<string>
END STRUCT

HashTable<Course> H            // key = courseNumber, value = Course
SET<string> Seen               // courseNumbers encountered
MAP<string, list<string>> PendingPrereqs

HELPERS
 FUNCTION SplitCSV(line : string) → list<string>
 FUNCTION IsBlank(line : string) → bool
 FUNCTION Trim(s : string) → string

PROCEDURE LoadCourses(filePath : string)
 OPEN filePath FOR reading AS f
 IF f failed THEN PRINT "Error: cannot open file"; RETURN

 lineNo ← 0
 WHILE NOT EOF(f)
```

```
line ← READLINE(f)
lineNo ← lineNo + 1
IF IsBlank(line) THEN CONTINUE

tokens ← SplitCSV(line)              // [courseNumber, name, prereq...]
IF SIZE(tokens) < 2 THEN
  PRINT "Format error on line ", lineNo, ": fewer than 2 fields"
  CONTINUE
END IF

courseNum ← Trim(tokens[0])
name      ← Trim(tokens[1])

IF courseNum = "" OR name = "" THEN
  PRINT "Format error on line ", lineNo, ": missing course number or name"
  CONTINUE
END IF

IF courseNum IN Seen THEN
  PRINT "Duplicate course ", courseNum, " on line ", lineNo
  CONTINUE
END IF

c ← NEW Course
c.courseNumber ← courseNum
```

```
    c.name        ← name
    c.prerequisites← EMPTY LIST
    H.Insert(courseNum, c)
    ADD courseNum TO Seen

    raw ← EMPTY LIST
    FOR i FROM 2 TO SIZE(tokens)-1
      p ← Trim(tokens[i])
      IF p ≠ "" THEN APPEND p TO raw
    END FOR
    PendingPrereqs[courseNum] ← raw
  END WHILE
  CLOSE f

  // Validate prerequisites and attach only those that exist
  FOR EACH (courseNum, rawList) IN PendingPrereqs
    FOR EACH p IN rawList
      IF p NOT IN Seen THEN
        PRINT "Prerequisite ", p, " for ", courseNum, " not defined → skipping"
        CONTINUE
      END IF
      c ← H.Search(courseNum)
      APPEND p TO c.prerequisites
      H.Insert(courseNum, c)  // upsert if needed
    END FOR
```

```
  END FOR
END PROCEDURE

PROCEDURE PrintCourseInfo(courseNum : string)
  c ← H.Search(courseNum)
  IF c NOT FOUND THEN PRINT "Course not found"; RETURN
  PRINT c.courseNumber, ", ", c.name
  IF SIZE(c.prerequisites) = 0 THEN
    PRINT "Prerequisites: None"
  ELSE
    SORT c.prerequisites ASC
    PRINT "Prerequisites: " + JOIN(c.prerequisites, ", ")
  END IF
END PROCEDURE

PROCEDURE PrintAllCourses()
  L ← H.GetAllValues()
  SORT L BY courseNumber ASC
  FOR EACH c IN L
    PRINT c.courseNumber, ", ", c.name
  END FOR
```

Tree Pseudocode

1. File Input Pseudocode

```
OPEN "courses.txt" as input file
IF file cannot be opened
   PRINT "Error: Cannot open file"
   EXIT program
END IF

FOR each line in file
   SPLIT line by commas into tokens
   courseNumber ← tokens[0]
   name ← tokens[1]
   prerequisites ← remaining tokens after index 1

   IF number of tokens < 2
      PRINT "Error: Line missing required parameters"
      CONTINUE to next line
   END IF

   CREATE new Course object
   Course.courseNumber ← courseNumber
   Course.name ← name
   Course.prerequisites ← prerequisites

   INSERT Course into Binary Search Tree
END FOR
```

```
CLOSE file
PRINT "All courses successfully loaded."
```

2. Validation Pseudocode

```
FOR each Course in the tree
   FOR each prerequisite in Course.prerequisites
      IF prerequisite not found in the tree
         PRINT "Error: prerequisite " + prerequisite + " does not exist as a course."
      END IF
   END FOR
END FOR
```

3. Course Object Pseudocode

```
STRUCT Course
   String courseNumber
   String name
   List<String> prerequisites
END STRUCT

STRUCT Node
   Course course
   Node left
   Node right
```

```
END STRUCT


CLASS BinarySearchTree
  Node root


  FUNCTION Insert(Course c)
    IF root = null
      root ← new Node(c)
    ELSE
      CALL addNode(root, c)
    END IF
  END FUNCTION


  FUNCTION addNode(Node node, Course c)
    IF c.courseNumber < node.course.courseNumber
      IF node.left = null
        node.left ← new Node(c)
      ELSE
        addNode(node.left, c)
      END IF
    ELSE IF c.courseNumber > node.course.courseNumber
      IF node.right = null
        node.right ← new Node(c)
      ELSE
        addNode(node.right, c)
```

```
        END IF
      END IF
    END FUNCTION
END CLASS
```

4. Print Course Information Pseudocode

```
FUNCTION printCourse(Tree<Course> courses, String courseNumber)
    node ← FIND node in tree with matching courseNumber
    IF node = null
        PRINT "Course not found."
        RETURN
    END IF

    PRINT node.course.courseNumber + ", " + node.course.name

    IF node.course.prerequisites is empty
        PRINT "No prerequisites"
    ELSE
        PRINT "Prerequisites: "
        FOR each prereq in node.course.prerequisites
            PRINT prereq
        END FOR
    END IF
END FUNCTION
```

5. Print All Courses (In-Order Traversal)

```
FUNCTION printAllCourses(Node node)
  IF node = null
    RETURN
  END IF
  printAllCourses(node.left)
  PRINT node.course.courseNumber + ", " + node.course.name
  printAllCourses(node.right)
```

Menu Pseudocode

```
MENU:
1. Load Data Structure
2. Print Course List
3. Print Specific Course
9. Exit Program
```

Runtime Analysis

Operation	Vector	Hash Table	Binary Search Tree
Insert	O(1) avg / O(n) worst	O(1) avg	O(log n) avg / O(n) worst
Search	O(n)	O(1) avg	O(log n) avg / O(n) worst
Sort	O(n log n)	N/A	O(n) via inorder traversal
Print	O(n)	O(n)	O(n)
Memory	Moderate	Higher (due to hashing)	Moderate

Advantages and Disadvantages

Vector: Simple implementation; good for sequential access and small data sets. Searching and inserting are slow (O(n)); must re-sort for ordered output.

Hash Table: Extremely fast lookups and inserts (O(1) average). No inherent order; higher memory usage; collision management required.

Binary Search Tree: Maintains sorted order automatically; efficient search and traversal. Can degrade to O(n) if unbalanced; more complex to implement.

Recommendation

After analyzing runtime and memory performance, the Binary Search Tree is the most suitable structure for ABCU's advising program. It maintains data in sorted order, enabling efficient course lookup and automatic alphabetical printing without separate sorting. While a hash table provides faster lookups, it lacks ordering, and vectors require repeated sorting. Therefore, the BST offers the best overall efficiency and aligns with the advisors' needs for quick searching and ordered output.