Mark Vincent Francisco
CS-300 - DSA: Analysis and Design
Computer Science Undergraduate Student
August 5th, 2025

Project One: Pseudocode and Runtime Analysis

Vector List Pseudocode

1. Design pseudocode to define how the program opens the file, reads the data from the file, parses each line, and checks for file format errors.

```
START

    INCLUDE fstream, sstream, string, vector

    DECLARE fileInput of type ifstream
    DECLARE line of type string
    DECLARE allCourses of type vector

    OPEN “courses.txt” using fileInput

    IF fileInput is open THEN
        OUTPUT “File opened successfully.”
        WHILE there are more lines to read in “courses.txt”
            READ line from fileInput

            IF line is empty THEN
                CONTINUE

            DECLARE ss as stringStream(line)
            DECLARE part of type string string
            DECLARE parts as vector of type string

            WHILE GETLINE ss into part using ‘,‘
                ADD part to parts
            END WHILE

            IF parts size is < 2 THEN
                OUTPUT “Error, line does not have course number and title.”
```

```
                    CONTINUE
                END IF

                APPEND line to allCourses

        ELSE
            OUTPUT "unable to open file."
        END IF

        CLOSE fileInput

END
```

2. Design pseudocode to show how to create course objects and store them in the appropriate data structure.

```
START
    Course Class
        DECLARE courseNumber of type string
        DECLARE title of type string
        DECLARE prerequisites of type vector<string>

        Constructor METHOD (courseNumber, Title, Prerequisites)
            This.courseNumber = courseNumber
            This.title = title
            This Prerequisites = prerequisites
        END METHOD

    END CLASS

    Main METHOD
        DECLARE allCourses of type Course

        DECLARE mathCourse as type Course
        MathCourse = new Course(MAT150, "Discrete Math", STAT200)

        APPEND mathCourse to allCourses
    END METHOD

END
```

3. Design pseudocode that will search the data structure for a specific course and print out course information and prerequisites.

START

```
Main METHOD
        DECLARE allCourses as vector of Course

        DECLARE mathCourse as type Course
        MathCourse = new Course(MAT150, “Discrete Math”, “STAT200”)

        APPEND mathCourse to allCourses

        DECLARE searchCourseNumber as string
        PRINT “Enter course number to search:”
        INPUT searchCourseNumber

        DECLARE found as boolean = false

        FOR each course in allCourses
                IF courseNumber == searchCourseNumber THEN
                        OUTPUT “Found a match!”
                        OUTPUT “Course Number:” + Course.courseNumber
                        OUTPUT “Title:” + course.title

                        IF course.prerequisites is not empty THEN
                                OUTPUT “Prerequisites:”
                                FOR each pr in course.prerequisites
                                        OUTPUT “- “ + pr
                                END FOR
                        ELSE
                                OUTPUT “Prerequisites: NONE”
                        END IF

                        SET found = TRUE
                        BREAK
                END IF
        END FOR

        IF found == FALSE THEN
```

```
            OUTPUT “Course not found.”
        END IF
    END METHOD
```

END

4. Create pseudocode for a menu

START

```
    DECLARE menuSelection as INT

    OUTPUT “MAIN MENU” newline
    OUTPUT “1. Load Course Data into Vector” newline
    OUTPUT “2. Print All Courses in Alphanumerical Order” newline
    OUTPUT “3. ”Search and Display Course” new line
    OUTPUT “9. Exit Program”
    OUTPUT “Select an Option: ” newline
    INPUT menuSelection

    WHILE menuSelection does not equal 9
        IF menuSelection equals 1
            RUN loadCoursesFromFile("courses.txt")
            OUTPUT "Course data loaded."
        ELSE IF menuSelection equals 2
            RUN sortCoursesAlphanumerically()
            RUN printCourses()
        ELSE IF menuSelection equals 3
            PRINT "Enter course number to search:"
            INPUT courseNum
            RUN searchCourse(courseNum)
        ELSE IF menuSelection equals 9
            OUTPUT "Goodbye."
            RETURN
        ELSE
            OUTPUT "Sorry, invalid option. Please try again."
        END IF
    END WHILE
```

END

5. Design pseudocode that will print out the list of the courses in the Computer Science program in alphanumeric order

START

```
FUNCTION sortCoursesAlphanumerically()
        SORT allCourses by courseNumber in ascending order
END FUNCTION
```

END

Vector Evaluation and Runtime Analysis

Evaluate the run time and memory of data structures that could be used to address the requirements

In implementing a vector for the purpose of storing and retrieving course information along with its prerequisites, the worst case runtime complexity in various operations such as: insert, delete, search, and printing represents O(n). This means that the runtime in any of these operations are directly proportional to the number of elements stored within the data structure; more elements, more time spent to process. However, parsing of each line contained within the fileInput requires a while loop inside of a while loop which would drastically increase the runtime depending on the number of prerequisites each course contains. The worst case runtime complexity in this case is $O(n^2)$.
In terms of memory storage in the vector data structure, the space complexity is O(n) as the vector does not require additional memory, especially when compared to the linked list data structure.

Code	Line Cost	# Times Executes	Total Cost
for all courses	1	n	n
if the course is the same as courseNumber	1	n	n
for each prerequisite of the course	1	1	1
for each prerequisite of the course	1	n	n
print the prerequisite course information	1	n	n*n
Total Cost	4n + 1		
Runtime	$O(n)$		

Hash Table Pseudocode

1. Design pseudocode to define how the program opens the file, reads the data from the file, parses each line, and checks for file format errors

```
START
        DECLARE allCourses as Array<String>
        DECLARE wholeLines as Array <String>

        OPEN “Courses.txt” file
        IF “courses.txt” cannot be opened
                OUTPUT “Error, cannot open file”
                RETURN
        END IF

        DECLARE “line” as string

        WHILE “Courses.txt” has more lines to read
                READ “line” from “Courses.txt”
                SPLIT “line” by ‘,’ into “parts”

                IF length of “parts” < 2
                        OUTPUT “Sorry, “line” must have at least 2 parameters.”
                        RETURN
                END IF

                ADD “line” INTO wholeLines
                ADD parts[0] INTO allCourses
        END WHILE

        CLOSE “Courses.txt”
END
```

2. Design pseudocode to show how to create course objects and store them in the appropriate data structure

```
START
    DECLARE hashTable "coursesData"

    FOR line IN wholeLines
        SPLIT line by ',' INTO parts

        courseNumber = Parts[0]
        courseTitle = Parts[1]
        coursePrerequisites = Parts[2:]



        FOR prerequisite in coursePrerequisites
            IF prerequisite does not exist in any courseNumber of courseData
                OUTPUT "Sorry, prerequisite does not exist for
courseNumber"
                RETURN
            END IF
        END FOR

        DECLARE courseObject
            SET courseObject.number = courseNumber
            SET courseObject.title = courseTitle
            SET courseObject.prerequisites = coursePrerequisites

        INSERT courseObject INTO coursesData USING courseNumber AS key
    END FOR
END
```

3. Design pseudocode that will print out course information and prerequisites

```
START
        METHOD printCourseInformation (coursesData)
            OUTPUT "All courses:"

            FOR CourseNumber, courseObject IN courseData
                OUTPUT courseObject.number + " - " + courseObject.title

                IF courseObject.prerequisites is NULL
                    OUTPUT "NO PREREQUISITES"
                END IF

                ELSE
                    OUTPUT "PREREQUISITES"
                    FOR prerequisite in courseObject.prerequisities
                        OUTPUT prerequisite + ", "
                    END FOR
                END ELSE

                OUTPUT newline
            END FOR
        END METHOD
    END
```

4. Create pseudocode for a menu

```
START

    DECLARE menuSelection as INT

    OUTPUT "MAIN MENU" newline
    OUTPUT "1. Load Course Data into Hash Table" newline
    OUTPUT "2. Print All Courses in Alphanumerical Order" newline
    OUTPUT "3. "Search and Display Course" new line
    OUTPUT "9. Exit Program"
    OUTPUT "Select an Option: " newline
    INPUT menuSelection

    WHILE menuSelection does not equal 9
```

```
        IF menuSelection equals 1
            RUN loadCoursesFromFile("courses.txt")
            OUTPUT "Course data loaded."
        ELSE IF menuSelection equals 2
            RUN sortCoursesAlphanumerically()
            RUN printCourses()
        ELSE IF menuSelection equals 3
            PRINT "Enter course number to search:"
            INPUT courseNum
            RUN searchCourse(courseNum)
        ELSE IF menuSelection equals 9
            OUTPUT "Goodbye."
            RETURN
        ELSE
            OUTPUT "Sorry, invalid option. Please try again."
        END IF
    END WHILE

END
```

5. Design pseudocode that will print out the list of the courses in the Computer Science program in alphanumeric order

```
FUNCTION sortCoursesAlphanumerically()

    EXTRACT all keys from hashTable into courseKeysList
        SORT courseKeysList in ascending order
    FOR each key in courseKeysList
        PRINT courseData[key]
    END FOR

END FUNCTION
```

Hash Table Evaluation and Runtime Analysis

The runtime complexity for the hash table data structure while inserting, deleting, or searching is on average O(1). However, when collisions occur (attempting to add multiple elements to the same bucket), this is when the runtime complexity could potentially increase to O(n) (worse case) when there are attempts to add multiple elements to a hash table with the same key. When executing the print function, the runtime complexity to print a single line would also be O(1). To be able to print all elements contained within a hash table, a for-loop nested within a for-loop is implemented; thus will affect the runtime complexity to $O(n^2)$ in order to reach every node.

The space complexity for the hash table is O(n) as the hash table does not require additional space beside the allocated memory to compute the key and return the value.

Code	Line Cost	# Times Executes	Total Cost
INSERT course object using courseNumber as key	**1**	**n**	**n**
SEARCH course by key (hash lookup)	**1**	**1**	**1**
OUTPUT match details	**1**	**1**	**1**
for each prerequisite in course.prerequisites	**1**	**m**	**m**
OUTPUT each prerequisite	**1**	**m**	**m**
Total Cost	**N + m + 2**		
Runtime	**Avg case: O(1); worst case for collisions: O(n); worst base to print the prerequisites: O(m)**		

Binary Tree Pseudocode

1. Design pseudocode to define how the program opens the file, reads the data from the file, parses each line, and checks for file format errors

```
START

    FUNCTION loadCoursesFromFile(filepath)

        DECLARE courseLines as list
        DECLARE courseNumbers as array
        DECLARE courseData as dictionary

        OPEN “Courses.txt” file
        IF “Courses.txt” cannot be opened
            OUTPUT “Error, cannot open file.”
            RETURN
        END IF

        FOR each line in “Courses.txt”
            READ line from “Courses.txt”
            IF line is empty
                CONTINUE to next line
            END IF

            SPLIT “line” by ‘,’ into “parts”
            IF number of “parts” < 2
                OUTPUT “Error, line must have at least 2 parameters”
                SKIP line
            END IF

            DECLARE courseNumber = parts[0]
            DECLARE courseTitle = parts[1]
            DECLARE prerequisites as list


            IF number of parts > 2
                FOR each part in parts[2:]
                    ADD part to prerequisites
                END FOR
```

```
            END IF

            ADD line to courseLines
            ADD courseNumber to courseNumbers
            ADD (courseTitle, prerequisites) to courseData[courseNumber]
        END FOR
```

2. Design pseudocode to show how to create course objects and store them in the appropriate data structure

CONTINUED…

```
    DECLARE Binary Search Tree "bstCourses"

    FOR each courseNumber in courseData
        GET (courseTitle, prerequisites) from courseData[courseNumber]
        CREATE "Course" object with courseNumber, courseTitle, prerequisites
        CALL bstCourses.insert(Course)
        RETURN bst
END FUNCTION
```

END

3. Design pseudocode that will print out course information and prerequisites

START

```
    FUNCTION printCoursesInOrder(node)
        IF node is NULL
            RETURN
        END IF

        CALL printCoursesInOrder traversing left subtree
            OUTPUT courseNumber, newline
            OUTPUT courseTitle, newline

            IF prerequisites NOT exists
                OUTPUT prerequisites, newline
            ELSE
                OUTPUT "No prerequisites for this course"
```

```
                    END IF

                    CALL printCoursesInOrder traversing right subtree
            END FUNCTION
END
```

4. Create pseudocode for a menu

```
START

    DECLARE menuSelection as INT

    OUTPUT “MAIN MENU” newline
    OUTPUT “1. Load Course Data into Binary Tree” newline
    OUTPUT “2. Print All Courses in Alphanumerical Order” newline
    OUTPUT “3. ”Search and Display Course” new line
    OUTPUT “9. Exit Program”
    OUTPUT “Select an Option: ” newline
    INPUT menuSelection

    WHILE menuSelection does not equal 9
        IF menuSelection equals 1
            RUN loadCoursesFromFile("courses.txt")
            OUTPUT "Course data loaded."
        ELSE IF menuSelection equals 2
            RUN sortCoursesAlphanumerically()
            RUN printCourses()
        ELSE IF menuSelection equals 3
            PRINT "Enter course number to search:"
            INPUT courseNum
            RUN searchCourse(courseNum)
        ELSE IF menuSelection equals 9
            OUTPUT "Goodbye."
            RETURN
        ELSE
            OUTPUT "Sorry, invalid option. Please try again."
        END IF
    END WHILE
```

END

5. Design pseudocode that will print out the list of the courses in the Computer Science program in alphanumeric order

START

```
FUNCTION sortCoursesAlphanumerically()
        RUN inOrderTraversal(root)
END FUNCTION

FUNCTION inOrderTraversal(node)
        IF node is NULL
                RETURN
        END IF
        RUN inOrderTraversal(node.left)
        OUTPUT courseNumber of Node
        OUTPUT title of Node
        RUN inOrderTraversal(node.right)
END FUNCTION
```

END

Binary Tree Evaluation and Runtime Analysis

The runtime complexity of the binary tree data structure for search, delete and insert operations is $O(log\ N)$ on average. The reason for fast execution of these tree operations is because the tree splits the search space whenever it moves down a level. The worst case run time complexity of a binary tree would be if the tree is skewed on one side where all nodes have one child, which would result in O(n). In this scenario, it would require the operation to traverse all nodes that exist in the tree.
The space complexity of the binary tree is O(n) which accounts for each node from the input.

Code	Line Cost	# Times Executes	Total Cost
INSERT course into BST	**1**	**n**	**n*O(log N)**
SEARCH course by courseNumber	O(Log N)	1	O(log N)
OUTPUT match details	1	1	1
FOR each prerequisite in course.prerequisites	1	m	m
OUTPUT each prerequisite	1	m	m
IN-ORDER traversal to print all courses	1	n	n
Total Cost	~nlog n + m + n		
Runtime	O(log N) avg. search/insert; O(n) to print all; O(m) to print prerequisites		

Advantages and Disadvantages of each Structure

Advantages of Vectors

- Straight forward implementations in code.
- Has direct indexing for each element contained within a vector.
- Can be easily ordered.

Disadvantages of Vectors

- Insertion or deletion from the middle, which could potentially cause half of the elements to shift an index.
- Slow when attempting to search for an element towards the end of a vector.

Advantages of Hash Tables

- Fastest of all three data structures with a runtime complexity of O(1).
- Very efficient when performing insert, search, and delete operations.

Disadvantages of Hash Tables

- Performance degradation if there are too many collisions that occur.
- There is no inherent order when storing elements, which can negatively affect the attempt to order the elements alphanumerically.

Advantages of Trees

- The structure of a tree is naturally sorted from left to right.
- Reasonable search times for search, insert, and delete operations.

Disadvantages of Trees

- A skewed binary tree could significantly affect runtime complexity.
- Requires the use of additional pointers, which affects memory.

Recommendation

My recommendation in selecting a data structure for storing all of ABCU courses in alphanumerical order would be a binary search tree. The reason for this is because of its swift searching capabilities and its ordered nature upon insertion of an element. As long as there are no significant number of collisions that occur when inserting courses into a tree, the delete, insert, and search operations should turn out to be O(log N) on average, which is an excellent runtime.

Space complexity is O(n) which is the same as the hash table and vector data structures, which makes the binary tree neither better nor worse than the other data structures. The vector would be the worst data structure to implement for this use case because of how runtime can significantly be increased if inserting or deleting from the middle of the list, or searching for an element at the end of the list (provided that the list of courses is large).

References

Vahid, F., Lysecky, S., Wheatland, N., Siu, R., Lysecky, R., Edgcomb, A., & Yuen, J. (2019). *CS 300: Data Structures and Algorithms* [zyBook]. Zyante Inc. https://www.zybooks.com