Jael Andre

Project One

CS 300

Dr. Webb

02/24/2025

**Introduction**

This document outlines the complete pseudocode and analysis for the ABCU Course Advising System. The system allows academic advisors to:

1. Load course data from a file.
2. Print courses in alphanumeric order.
3. Display detailed course information, including prerequisites.

The analysis compares three data structures: **Vector**, **Hash Table**, and **Binary Search Tree (BST)**, concluding with a recommendation based on efficiency and runtime complexity.

**ABCU Course Advising System: Pseudocode and Runtime Analysis**

**Pseudocode for Course Advising System**

**Course Structure Definition:**

STRUCT Course

courseNumber: STRING

courseTitle: STRING

prerequisites: LIST OF STRING

**Menu Function:**

FUNCTION menu()

WHILE TRUE

PRINT "1: Load Data, 2: Print Course List, 3: Search Course, 9: Exit"

INPUT choice

SWITCH choice

CASE 1: CALL loadCourses()

CASE 2: CALL printSortedCourses()

CASE 3: CALL displayCourseInfo()

CASE 9: BREAK

**Vector Data Structure:**

FUNCTION loadCourses(filename, courses)

OPEN file FOR reading

WHILE NOT end of file

READ line FROM file

SPLIT line INTO tokens BY ','

IF LENGTH of tokens < 2

PRINT "Invalid format."

CONTINUE

CREATE course OBJECT

SET courseNumber = tokens[0], courseTitle = tokens[1], prerequisites = tokens[2:]

APPEND course TO courses

CLOSE file

**Hash Table Data Structure:**

FUNCTION loadCourses(filename, coursesHashTable)

OPEN file FOR reading

WHILE NOT end of file

READ line FROM file

SPLIT line INTO tokens BY ','

IF LENGTH of tokens < 2

PRINT "Invalid format."

CONTINUE

CREATE course OBJECT

SET courseNumber = tokens[0], courseTitle = tokens[1], prerequisites = tokens[2:]

INSERT course INTO coursesHashTable USING courseNumber AS KEY

**Binary Search Tree Data Structure:**

STRUCT TreeNode

course: Course

left: TreeNode

right: TreeNode

FUNCTION insertCourse(root, course)

IF root IS NULL

CREATE newNode WITH course

RETURN newNode

IF course.courseNumber < root.course.courseNumber

root.left = insertCourse(root.left, course)

ELSE IF course.courseNumber > root.course.courseNumber

root.right = insertCourse(root.right, course)

RETURN root

**Runtime Analysis**

|  |  |  |  |
| --- | --- | --- | --- |
| **Data Structure** | **Insertion Time** | **Search Time** | **Memory Useage** |
| **Vector** | O(1) (amortized) | O(n) | |  | | --- | | O(n) | |
| **Hash Table** | |  | | --- | | O(1) (average), O(n) (worst) |  |  | | --- | |  | | O(1) (average), O(n) (worst) | O(n) |
| **BST** | |  | | --- | | O(log n) (balanced), O(n) (worst) |  |  | | --- | |  | | |  | | --- | | O(log n) (balanced), O(n) (worst) |  |  | | --- | |  | | O(n) |

**Advantages and Disadvantages Analysis**

**Vector:**

* *Advantages:* Simple to implement, efficient memory usage.
* *Disadvantages:* Slow search and sorting times for large datasets (O(n)).

**Hash Table:**

* *Advantages:* Fast average-case insertion and search times (O(1)).
* *Disadvantages:* Potential for collisions, leading to O(n) worst-case runtime.

**Binary Search Tree (BST):**

* *Advantages:* Provides sorted data naturally through in-order traversal.
* *Disadvantages:* Can degrade to O(n) in worst cases if unbalanced.

**Recommendation**

The **Hash Table** is the best data structure for this system due to its quick average-case insertion and search times. Its efficiency significantly improves the speed of searching and accessing course data compared to Vectors or Binary Search Trees. Although the BST is effective for sorted data, it poses a higher risk of inefficiency if not balanced.