Nicholas Liousas

2/15/2025

CS 300

**CS 300 Project One**

**Part 1: Pseudocode**

**Vector Implementation:**

// Function to import and validate course data from a file

void importCourseData(Vector<Course> &courses, String fileName) {

OPEN fileName in read mode

IF file cannot be opened THEN

PRINT "Error: File cannot be opened."

EXIT program

END IF

WHILE not end of file DO

READ a line from the file

SPLIT the line into tokens using a comma as a delimiter

// Validate the line format

IF number of tokens < 2 THEN

PRINT "Error: Line does not have at least two parameters."

CONTINUE

END IF

// Extract course details

Course newCourse

newCourse.number = tokens[0]

newCourse.name = tokens[1]

// Parse prerequisites

FOR each token from the 3rd onward DO

ADD token to newCourse.prerequisites

END FOR

ADD newCourse to courses

END WHILE

CLOSE fileName

}

// Function to search for and print course information

void searchCourse(Vector<Course> courses, String courseNumber) {

FOR each course in courses DO

IF course.number == courseNumber THEN

PRINT course details and prerequisites

RETURN

END IF

END FOR

PRINT "Error: Course not found."

}

**Hash Table Implementation:**

// Function to import and validate course data into a hash table

void importCourseData(HashTable<Course> &courses, String fileName) {

OPEN fileName in read mode

IF file cannot be opened THEN

PRINT "Error: File cannot be opened."

EXIT program

END IF

WHILE not end of file DO

READ a line from the file

SPLIT the line into tokens using a comma as a delimiter

// Validate the line format

IF number of tokens < 2 THEN

PRINT "Error: Line format incorrect."

CONTINUE

END IF

Course newCourse

newCourse.number = tokens[0]

newCourse.name = tokens[1]

// Parse prerequisites

FOR each token from the 3rd onward DO

ADD token to newCourse.prerequisites

END FOR

INSERT newCourse INTO courses USING newCourse.number AS key

END WHILE

CLOSE fileName

}

// Function to search for and print course information using hash table

void searchCourse(HashTable<Course> courses, String courseNumber) {

IF courseNumber exists in courses THEN

PRINT course details and prerequisites

ELSE

PRINT "Error: Course not found."

END IF

}

**Binary Search Tree (BST) Implementation**

// Function to insert course into BST

void insertCourse(BST &courseTree, Course newCourse) {

IF courseTree is NULL THEN

courseTree = new BSTNode(newCourse)

ELSE IF newCourse.number < courseTree.number THEN

insertCourse(courseTree.left, newCourse)

ELSE

insertCourse(courseTree.right, newCourse)

END IF

}

// Function to search for and print course information

void searchCourse(BSTNode\* node, String courseNumber) {

IF node is NULL THEN

PRINT "Error: Course not found."

RETURN

END IF

IF courseNumber == node.data.number THEN

PRINT course details and prerequisites

ELSE IF courseNumber < node.data.number THEN

searchCourse(node.left, courseNumber)

ELSE

searchCourse(node.right, courseNumber)

END IF

}

**Part 2: Runtime Analysis**

| **Data structure** | **Reading files & Creating course objects** | **Searching for a course** | **Sorting Courses** |
| --- | --- | --- | --- |
| **Vector** | O(n) | O(n) | O(n log n) |
| **Hash Table** | O(n) | O(1) (avg), O(n) (worst case) | N/A (unordered) |
| **BST** | O(n log n) (if balanced) | O(log n) (if balanced) | O(n log n) |

**Part 3: Evaluation and Recommendation**

Vector Analysis

* Advantages: Simple to implement, maintains order.
* Disadvantages: Linear search is inefficient, sorting takes extra time.
* Best For: Small datasets where sorting is needed.

Hash Table Analysis

* Advantages: Fast retrieval (O(1) on average).
* Disadvantages: No natural sorting, worst-case lookup is O(n).
* Best For: Large datasets where fast lookups are required.

BST Analysis

* Advantages: Searches and inserts in O(log n) (if balanced), maintains order.
* Disadvantages: Requires balancing to avoid O(n) worst case.
* Best For: Applications requiring sorted data with efficient searching.

**Part 4: Recommendation**

The hash table is the best choice for this application because it provides fast lookups (O(1) on average), making it highly efficient for retrieving course information. This allows students and advisors to quickly search for courses and their prerequisites without scanning through large datasets. However, hash tables do not maintain a natural order, so sorting is required when listing all courses. Since searching is the most frequent operation in this program, the speed advantage of a hash table outweighs the need for sorting. Overall, due to its speed and efficiency, the hash table is the best choice for this application.