**Project One: Pseudocode and Runtime Analysis**

**Ryan E Shaffer**

**SNHU**

**CS-300: DSA Analysis and Design**

**Leslie Gruner**

**August 11th, 2024**

Project One: Pseudocode and Runtime Analysis

Pseudocode for Course Information and Prerequisites

1. Vector Implementation

// Function to load courses from file into a vector

void loadCourses(Vector<Course> courses, String filename) {

open file filename

while not end of file

read line from file

parse line into courseNumber, name, prerequisites

create a Course object with parsed data

add Course object to vector courses

end while

close file

}

// Function to print all courses in alphanumeric order

void printAllCourses(Vector<Course> courses) {

sort vector courses by courseNumber

for each course in courses

print courseNumber and name

end for

}

// Function to print a specific course and its prerequisites

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

for each course in courses

if courseNumber matches course.courseNumber

print course name

for each prerequisite in course.prerequisites

print prerequisite courseNumber

end for

end if

end for

}

2. Hash Table Implementation

// Function to load courses from file into a hash table

void loadCourses(HashTable<Course> courses, String filename) {

open file filename

while not end of file

read line from file

parse line into courseNumber, name, prerequisites

create a Course object with parsed data

add Course object to hash table courses with key as courseNumber

end while

close file

}

// Function to print all courses in alphanumeric order

void printAllCourses(HashTable<Course> courses) {

create a list from hash table keys (courseNumbers)

sort list by courseNumber

for each courseNumber in sorted list

retrieve course from hash table

print courseNumber and name

end for

}

// Function to print a specific course and its prerequisites

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

retrieve course from hash table using courseNumber

if course exists

print course name

for each prerequisite in course.prerequisites

print prerequisite courseNumber

end for

end if

}

3. Binary Search Tree Implementation

// Function to load courses from file into a binary search tree

void loadCourses(Tree<Course> courses, String filename) {

open file filename

while not end of file

read line from file

parse line into courseNumber, name, prerequisites

create a Course object with parsed data

add Course object to tree courses

end while

close file

}

// Function to print all courses in alphanumeric order

void printAllCourses(Tree<Course> courses) {

traverse tree in order (in-order traversal)

for each course in traversal

print courseNumber and name

end for

}

// Function to print a specific course and its prerequisites

void printCourseDetails(Tree<Course> courses, String courseNumber) {

find course in tree using courseNumber

if course exists

print course name

for each prerequisite in course.prerequisites

print prerequisite courseNumber

end for

end if

}

**Runtime Analysis**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Data Structure** | **Operation** | **Code Line Description** | **Line Cost** | **Times Executed** | **Total Cost** | **Complexity** |
| Vector | |  | | --- | | Load Courses |  |  | | --- | |  | | ‘for all courses’ | 1 | n | n | O(n) |
|  | Print All Courses | ‘sort vector courses by courseNumber’ | |  | | --- | | n log n |  |  | | --- | |  | | 1 | n log n | O(n log n) |
|  | Print Course Details | ‘for each prerequisite of the course’ | 1 | p (avg) | p | O(n + p) |
| Hash Table | Load Courses | ‘add Course object to hash table’ | 1 | n | n | O(n) |
|  | Print All Courses | sort list by courseNumber | n log n | 1 | n log n | O(n log n) |
|  | Print Course Details | for each prerequisite of the course | 1 | p (avg) | p | O(1 + p) |
| |  | | --- | | Binary Tree |  |  | | --- | |  | | Load Courses | add Course object to tree | log n | n | n log n | O(n log n) |
|  | Print All Courses | traverse tree in order | 1 | n | n | O(n) |
|  | Print Course Details | find course in tree | log n | 1 | log n | O(log n + p) |

**Analysis of Advantages and Disadvantages**

Vector:

Advantages: Easy to implement, simple to use for sequential access.

Disadvantages: Inefficient for large datasets when searching or inserting in the middle.

Hash Table:

Advantages: Fast access for search operations, O(1) average time.

Disadvantages: Inefficient for ordered data, higher memory usage, complexity in handling collisions.

Binary Search Tree:

Advantages: Efficient search and insertion with O(log n) time, maintains data in sorted order.

Disadvantages: More complex implementation, performance degrades to O(n) if not balanced.

**Recommendation**

Based on the Big O analysis, the Binary Search Tree is recommended. It provides efficient search and insertion while maintaining an ordered structure, making it ideal for the task of printing courses in alphanumeric order. Although the hash table offers O(1) search time, its inability to maintain order makes it less suitable for this task. The vector, while simple, becomes less efficient as the dataset grows. Therefore, the binary search tree strikes the best balance between performance and functionality for this project