**6-2: Project One**

Sharif Ayesh

Southern New Hampshire University

CS-300: Analysis and Design

Professor Adamo

April 7, 2025

**Project One: 6-2**

**Pseudocode for File Handling**

function loadFile(String filename) {

open file

while not end of file {

read line

parse line into courseNumber, courseTitle, prerequisites

if format is valid {

create a Course object with courseNumber, courseTitle, prerequisites

add the Course object to the data structure (e.g., vector, hash table, tree)

} else {

print "Error: Invalid format in line"

}

}

close file

}

**Course Object Creation**

function createCourse(String courseNumber, String courseTitle, List<String> prerequisites) {

Course newCourse

newCourse.courseNumber = courseNumber

newCourse.courseTitle = courseTitle

newCourse.prerequisites = prerequisites

return newCourse

}

**Pseudocode for Printing Course Info**

function searchCourse(DataStructure<Course> courses, String courseNumber) {

if data structure is not empty {

course = search for courseNumber

if course is found {

print course title and course number

for each prerequisite in course.prerequisites {

print prerequisite

}

} else {

print "Course not found"

}

}

}

**Pseudocode for Menu Options**

function displayMenu() {

print "1: Load course data"

print "2: Print list of all courses"

print "3: Print course and prerequisites"

print "9: Exit"

get user input

if input is 1 {

call loadFile function

} else if input is 2 {

call printAllCourses function

} else if input is 3 {

call searchCourse function

} else if input is 9 {

exit program

}

}

**Sorting and Printing Course List**

function printAllCourses(DataStructure<Course> courses) {

if using vector {

sort courses by courseNumber

} else if using tree {

traverse tree in-order

}

for each course in courses {

print course number and title

}

}

**Runtime Analysis**

|  |  |  |  |
| --- | --- | --- | --- |
| **Operation** | **Vector** | **Hash Table** | **Binary Search Tree** |
| **Load File** | O(n) | O(n) | |  | | --- | |  |   O(n) |
| **Search Course By Number** | O(n | O(1) | O(log n) |
| **Sort and Print Courses** | O(n log n) | O(n) | O(n) (in-order) |

**Recommendation**

After analyzing the performance of all three data structures, the Binary Search Tree (BST) is the best choice for this project. It meets both requirements from the academic advisors: fast access to course details and the ability to display all courses in alphanumeric order.

BSTs keep all data sorted as it’s added, which means we don’t need to sort it later. This makes them ideal for Option 2 in the menu, where users want to see a complete list of courses. Searching is also efficient, with the worst-case time of O(log n), which makes it fast even with a large number of courses.

Hash tables are faster for searching individual courses O(1), but they don’t keep the data sorted. To print a list in order, we’d need to do extra work, which adds complexity. Vectors are easy to use, but they are slower for searching, requiring O(n) time, and sorting every time the course list is printed, which takes O(n log n) time.

In summary, the BST is the best fit because it offers good performance for both searching and sorting without needing extra steps or structures. It’s efficient, organized, and scalable, making it the most suitable data structure for ABCU’s advising system.