Justin Hardin

CS 300 Data Structures and Algorithms: Analysis and Design

6-2 Project One

February 16, 2025

-------- Milestone 1 – Vector --------

// Course struct to hold course data

Struct Course {

declare String courseNumber

declare String courseName

declare Vector<String> prerequisites

}

// Validate and read course data from file

void loadCourseDataFromFile(String filename) {

open file

if file fails to open

display error message “File not found”

exit program

create empty vector to store courses (Cost: 1)

// Collect all valid course numbers first, so that later when checking for prerequisites,

// we know that they are valid course numbers

// Make sure that each line has at least two parameters

for each line in file

split line by comma

if line has less than two parameters

display error “Invalid course format: “ + line

exit program (Cost: 1)

add first parameter (course number) to vector

// Validate prerequisites and create courses

reset file to beginning

for each line in file

split line by comma

for each prerequisite in line // starting at the third parameter

if prerequisite not found in course number vector

display error “Invalid prerequisite: “ + prerequisite

exit program

close file

}

// Create Course objects and store in vector

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

open file

for each line in file

split line by comma

create new Course object

set course number to first parameter

set course name to second parameter

// Store prerequisites if they exist

for each remaining parameter in line

add parameter to course prerequisites list

add course object to courses vector

close file

}

// Search for and print course information

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

for all courses

if the course is the same as courseNumber

print out the course information

for each prerequisite of the course

print the prerequisite course information

}

-------- Milestone 2 – Hash Table --------

// Course struct to hold course data

Struct Course {

declare String courseNumber

declare String courseName

declare Vector<String> prerequisites

}

// Validate and read course data from file

Void loadCourseDataFromFile(String filename) {

open file

if file fails to open

display error message “File not found”

exit program

create empty hash table to store courses

create empty vector to store valid course numbers

// Collect all valid course numbers first, so that later when

// checking for prerequisites, we know that they are valid

// course numbers

for each line in file

split line by comma

if line has less than two parameters

display error “Invalid course formate: “ + line

exit program

add first parameter (course number) to vector

// Validate prerequisites

reset file to beginning

for each line in file

split line by comma

for each prerequisite in line (starting at third parameter)

if prerequisite not found in course number vector

display error “Invalid prerequisite: “ + prerequisite

exit program

close file

}

// Create Course objects and store in hash table

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

open file

for each line in file

split line by comma

create new Course object

set course number to first parameter

set course name to second parameter

// Store prerequisites if they exist

for each remaining parameter in line

add parameter to course prerequisites list

insert course object into courses hash table using course number as key (Cost: 1, Executions: n)

close file (Cost: 1, Executions: 1)

}

// Search for and print course information

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

retrieve course from courses hash table using courseNumber as key

if course exists

print course number and name

if prerequisites exist

for each prerequisite

retrieve prerequisite course from course hash table

print prerequisites course number and name

else

print “No prerequisites”

else

print “Course not found”

}

-------- Milestone 3 – Binary Search Tree --------

// Course struct to hold course data

Struct Course {

declare String courseNumber

declare String courseName

declare Vector<String> prerequisites

}

// Binary Search Tree Node struct

Struct Node {

declare Course course

declare Node\* left

declare Node\* right

}

// Function to create new node

Node\* createNode(Course course) {

create new Node

set node.course equal to course

set node.left to null

set node.right to null

return node

}

// Function to insert a node into the bst

Node\* insert(Node\* root, Course course) {

if root is null

return call createNode(course)

if course.courseNumber is less than root.courseNumber

set root.left equal to call insert(root.left, course)

else if course.courseNumber is greater than root.course.courseNumber

set root.right equal to call insert(root.right, course)

return root

}

// Validate and read course data from file

Void loadCourseDataFromFile(String filename) {

open file (Cost: 1)

if file fails to open

put error message “File not found” to output

exit program

create empty vector to store course numbers

create root node pointer and set to null

// Collect all valid course numbers

// Make sure each line has at least two parameters

for each line in file

split line by comma

if line has less than two parameters

put error message “Invalid course format: “ + line to output

exit program

add first parameter (the course number) to vector

// Validate prerequisites

reset file to beginning

for each line in file

split line by comma

for each prerequisite in line (starts at third parameter)

if prerequisite not found in course number vector

put error message “Invalid prerequisite: “ + prerequisite to output

exit program

close file

}

// Create Course objects and store in the bst

Node\* createAndStoreCourses(String filename) {

open file

create root node pointer and set to null

for each line in file

split line by comma

create new Course object

set course number to first parameter

set course name to second parameter

// Store prerequisites if they exist

for each remaining parameter in line

add parameter to course prerequisites list

// Insert course into the bst

set root equal to call insert(root, course)

close file

return root

}

// Helper function to find a course in the bst

Node\* findCourse(Node\* root, String courseNumber) {

if root is null or root.course.courseNumber equals courseNumber

return root

if courseNumber is less than root.course.courseNumber

return call findCourse(root.left, courseNumber)

else

return call findCourse(root.right, courseNumber)

}

// Search for and print course information

void searchCourse(Node\* root, String courseNumber) {

create courseNode and set equal to call findCourse(root, courseNumber)

if courseNode is null

put “Course not found” into output

return

// Output course info

put courseNode.course.courseNumber + courseNode.course.courseName into output

// Output prerequisites

if courseNode.course.prerequisites is not empty

put prerequisites into output

for each prerequisite in courseNode.course.prerequisites

create prerequisiteNode and set equal to call findCourse(root, prerequisite)

if prerequisiteNode is not null

put prerequisiteNode courseNumber + prerequisiteNode courseName into output

}

-------- Menu System --------

// Main Menu and program control

Void displayMenu() {

declare Boolean running

set running to true

declare courseData // Stores data structure (Vector, Hash Table, or Binary Search Tree

set courseData to null

while running is true

put “Welcome to the Course Management System” into output

put “1. Load Data Structure” into output

put “2. Print Course List” into output

put “3. Print Course Information” into output

put “9. Exit” into output

put “Enter your choice: “ into output”

input choice from user

if choice equals 1

put “Enter filename: “ into output”

input filename from user

// Validate data

Call loadCourseDataFromFile(filename)

// Then create and store courses

set courseData equal to call createAndStoreCourses(filename)

put “Data loaded successfully” into output

else if choice equals 2

if courseData equals null

put “Please load data first” into output

else

call printSortedCourseList(courseData)

else if choice equals 3

if courseData equals null

put “Please load data first” into output

else

put “Enter course number: “ into output

input courseNumber from user

call searchCourse(courseData, courseNumber)

else if choice equals 9

set running to false

else

put “Invalid option. Please try again.” into output

}

-------- Print Sorted Course List For Each Data Structure --------

// Using Vector data structure

Void printSortedCourseList(Vector<Course> courses) {

if courses is empty

put “No courses to display” into output

return

// Create temporary vector for sorting

declare Vector<Course> sortedCourses

set sortedCourses equal to courses

// Sort courses by course number

declare Integer n

set n equal to length of sortedCourses

for i equals 0 to n-1

for j equals 0 to n-i-1

// Compare adjacent course numbers

if sortedCourses[j].courseNumber is greater than sortedCourses[j+1].courseNumber

// Swap courses

declare Course temp

set temp equal to sortedCourses[j]

set sortedCourses[j] equal to sortedCourses[j+1]

set sortedCourses[j+1] equal to temp

// Print all courses

for each course in sortedCourses

put course.courseNumber + “, “ + course.courseName into output

}

// Using Hash Table data structure

Void printSortedCourseList(HashTable<Course> courses) {

if courses is empty

put “No courses to display” into output

return

// Create temporary vector for sorting

declare Vector<Course> sortedCourses

// Copy all courses from hash table to vector

for each entry in courses

add entry.value to sortedCourses

// Sort courses by course number

declare Integer n

set n equal to length of sortedCourses

for i equals 0 to n-c

// Find minimum course number in unsorted portion

declare Integer minIndex

set minIndex equal to 1

for j equals i+1 to n-1

if sortedCourses[j].courseNumber less than sortedCourses[minIndex].courseNumber

set minIndex equal to j

// Swap minimum with first unsorted position

If minIndex not equal to i

declare Course temp

set temp equal to sortedCourses[i]

set sortedCourses[i] equal to sortedCourses[minIndex]

set sortedCourses[minIndex] equal to temp

// Print all courses

for each course in sortedCourses

put course.courseNumber + “, “ + course.courseName into output

}

// Using Binary Search Tree data structure

Void printSortedCourseList(Node\* root) {

// Use inorder traversal for sorted output

if root equals null

return

// Recursively traverse left subtree

call printSortedCourseList(root.left)

// Print current node

cut root.course.courseNumber + “, “ root.course.courseName into output

// Recursively traverse right subtree

call printSortedCourseList(root.right)

}

-------- Runtime Analysis --------

A screenshot of a table

AI-generated content may be incorrect.

-------- Recommendation --------

Based on the findings in the runtime analysis, I’d like to recommend using the hash table implementation. It has the best runtime of the three at O(n) because of its constant time search for prerequisites. It has a runtime of O(1) for looking up individual courses. Since the dataset of university courses is relatively small, the higher memory overhead shouldn’t be an issue. It’s implementation will be more difficult than a vector, but less so than a binary tree structure, making for a good balance between ease of implementation and performance.