Gregory Anderson

CS300

8/15/2025

Project One

**1. Resubmit pseudocode from previous pseudocode assignments and update as necessary**.

**Vector Data Structure**

// Define Course Struct

struct Course:

*number* : String

*title* : String

*prereqs* : Vector<String>

// Load and Validate File

function loadCourses(*filename*):

open the file with the given *filename* for reading

if the file cannot be opened:

print "Error: Cannot open file"

exit the program

create empty vector<Course> called *courses*

for each line in the file:

if the line is empty:

continue to the next line

split the line by comma into a list called *values*

if the length of *values* is less than 2:

print "Error: Invalid line format"

continue to the next line

*number* = *values*[0]

*title* = *values*[1]

*prereqs* = empty list

for i from 2 to length of *values* - 1:

add *values*[i] to *prereqs*

create a new Course object with *number, title, prereqs*

add Course to *courses* vector

// Validate that all prerequisites refer to existing courses

for each *course* in *courses*:

for each *prereq* in *course.prereqs*:

*found* = false

for each *c* in *courses*:

if *c.number* == *prereq*:

*found* = true

break

if *found* == false:

print "Error: Prerequisite " + *prereq* + " not found for course " + *course.number*

// Search for specific course and print course info

function searchCourse(*courses*, *targetNumber*):

for each *course* in *courses*:

if *course.number* == *targetNumber*:

print *course.number* + ", " + *course.title*

if *course.prereqs* is empty:

print "No prerequisites"

else:

print "Prerequisites:"

for each *prereqNumber* in *course.prereqs*:

print *prereqNumber*

return

print "Course " + targetNumber + " not found"

**Hash Table Data Structure**

// Define Course Struct

struct Course:

number : String

title : String

prereqs : List<String>

// Define Hash Table

class HashTable:

size : Integer

table : Array of Linked Lists of Course

function hash(key):

return integer value of key mod size

function insert(course):

key = hash(course.number)

insert course at head of linked list at table[key]

function search(courseNumber):

key = hash(courseNumber)

traverse linked list at table[key]

return course if match found

function printAll():

for each linked list in table:

for each course in list:

print course.number, course.title, and course.prereqs

// Load and Validate File

function loadCourses(filename):

open file for reading

if file fails to open:

print error message and exit

create empty HashTable called courses

create empty Set<String> for courseNumbers

create empty List<Course> for validationQueue

for each line in file:

trim whitespace

if line is empty:

skip to next

split line by comma into list called tokens

if length of tokens < 2:

print format error and continue

number = tokens[0]

title = tokens[1]

prereqs = tokens from index 2 onward

create Course object with number, title, and prereqs

insert course into hash table

add number to courseNumbers set

add course to validationQueue

// Validate prerequisites

for each course in validationQueue:

for each prereq in course.prereqs:

if prereq not in courseNumbers:

print error: "Missing prerequisite definition for " + prereq

return courses

**Binary Search Tree Data Structure**

// Define Course struct

struct Course:

number : String

title : String

prereqs : List<String>

// Define BST Node struct

struct Node:

course : Course

left : Node

right : Node

// Define Binary Search Tree

class BST:

root : Node

function insert(course):

if root is null:

root = new Node(course)

else:

insertNode(root, course)

function insertNode(node, course):

if course.number < node.course.number:

if node.left is null:

node.left = new Node(course)

else:

insertNode(node.left, course)

else:

if node.right is null:

node.right = new Node(course)

else:

insertNode(node.right, course)

function search(courseNumber) -> Course or null:

current = root

while current is not null:

if courseNumber == current.course.number:

return current.course

else if courseNumber < current.course.number:

current = current.left

else:

current = current.right

return null

function inOrderTraversal(node):

if node is null: return

inOrderTraversal(node.left)

displayCourse(node.course)

inOrderTraversal(node.right)

// Main file loading function

function loadCourses(filename, bst):

open file with filename for reading

if file cannot be opened:

print "Error: Cannot open file"

exit program

allCourses = empty list // temporarily store for prerequisite validation

for each line in file:

trim whitespace

if line is empty: continue

split line by comma into values

if length of values < 2:

print "Error: Invalid line format"

continue

number = values[0]

title = values[1]

prereqs = empty list

for i from 2 to length(values)-1:

add values[i] to prereqs

course = new Course(number, title, prereqs)

add course to allCourses list

bst.insert(course)

// Validate prerequisites

for each course in allCourses:

for each prereq in course.prereqs:

if bst.search(prereq) is null:

print "Error: Prerequisite " + prereq + " not found in file"

// Display a course and its prerequisites

function displayCourse(course):

print course.number + ", " + course.title

if course.prereqs is empty:

print "Prerequisites: None"

else:

print "Prerequisites: " + join(course.prereqs, ", ")

// Search for a specific course

function searchCourse(bst, courseNumber):

course = bst.search(courseNumber)

if course is null:

print "Course " + courseNumber + " not found"

else:

displayCourse(course)

**2. Create pseudocode for a menu**.

function mainMenu():

while True:

print "1. Load course data"

print "2. Print course list (alphanumeric order)"

print "3. Print course information"

print "9. Exit"

choice = prompt user for menu option

if choice == 1:

filename = ask user for filename

ds = loadCourses\_<chosen DS>(filename)

else if choice == 2:

if ds is empty:

print "Error: No data loaded. Please load course file first."

else:

printCourseList\_<DS>(ds)

else if choice == 3:

if ds is empty:

print "Error: No data loaded. Please load course file first."

else:

courseNumber = ask user for course number

searchCourse\_<DS>(ds, courseNumber)

else if choice == 9:

print "Exiting program."

break

else:

print "Invalid selection. Please choose again."

**3.** **Design pseudocode that will print out the list of the courses in the Computer Science program in alphanumeric order.**

**Vector Data Structure:**function printCourseList\_Vector(courses):

if courses is empty: print "No data"; return

sort courses by course.number ascending (alphanumeric)

for c in courses:

print c.number + ": " + c.title

**Hash Table Data Structure:**

function printCourseList\_Hash(table):

list = empty list

for each bucket in table:

for each course in bucket:

list.append(course)

sort list by course.number ascending

for c in list:

print c.number + ": " + c.title

**BST Data Structure:**

function printCourseList\_BST(tree):

inOrder(tree.root)

function inOrder(node):

if node is null: return

inOrder(node.left)

print node.course.number + ": " + node.course.title

inOrder(node.right)

**4. Evaluate the run time and memory of data structures that could be used to address the requirements**.

**Vector**

| Code | Line Cost | # Times Executes | Total Cost |
| --- | --- | --- | --- |
| Open the file with the given filename for reading | 1 | 1 | 1 |
| If the file cannot be opened: print error and exit | 1 | 1 | 1 |
| Create empty vector<Course> called courses | 1 | 1 | 1 |
| For each line in the file | 1 | n | n |
| If the line is empty:continue | 1 | n | n |
| split the line by comma into a list called values | 1 | n | n |
| If the length of values is less than 2: print error and continue | 1 | n | n |
| number = values[0]; title = values[1] | 1 | n | n |
| prereqs = empty list | 1 | n | n |
| for i from 2 to length(values) - 1: add values[i] to prereqs | 1 | n | n |
| create a new Course object (number, title, prereqs) | 1 | n | n |
| append course to courses vector | 1 | n | n |
| Total Cost | | | 9n + 3 |
| Runtime | | | O(n) |

**Hash Table:**

| Code | Line Cost | # Times Executes | Total Cost |
| --- | --- | --- | --- |
| open the file with the given filename for reading | 1 | 1 | 1 |
| if the file cannot be opened: print error and exit | 1 | 1 | 1 |
| create empty hash table<Course> called courses | 1 | 1 | 1 |
| for each line in the file | 1 | n | n |
| trim leading/trailing whitespace from the line | 1 | n | n |
| if the line is empty: continue to the next line | 1 | n | n |
| split the line by comma into a list called values | 1 | n | n |
| if the length of values is less than 2: print error and continue | 1 | n | n |
| number = values[0] | 1 | n | n |
| title = values[1] | 1 | n | n |
| prereqs = empty list | 1 | n | n |
| for each prerequisite in values | 1 | n | n |
| add prerequisite to prereqs | 1 | n | n |
| create new Course object(number, title, prereqs) | 1 | n | n |
| insert Course object into hash table | 1 | n | n |
| Total Cost | | | 12n + 3 |
| Runtime | | | O(n) |

**BST Data Structure:**

| Code | Line Cost | # Times Executes | Total Cost |
| --- | --- | --- | --- |
| open the file with the given filename for reading | 1 | 1 | 1 |
| if the file cannot be opened: print error and exit | 1 | 1 | 1 |
| create empty binary search tree<Course> called bst | 1 | 1 | 1 |
| for each line in the file | 1 | n | n |
| trim leading/trailing whitespace from the line | 1 | n | n |
| if the line is empty: continue to the next line | 1 | n | n |
| split the line by comma into a list called values | 1 | n | n |
| if the length of values is less than 2: print error and continue | 1 | n | n |
| number = values[0] | 1 | n | n |
| title = values[1] | 1 | n | n |
| prereqs = empty list | 1 | n | n |
| for each prerequisite in values | 1 | n | n |
| add prerequisite to prereqs | 1 | n | n |
| create new Course object (number, title, prereqs) | 1 | n | n |
| insert Course object into BST | n | n | n |
| Total Cost | | | n2+11n + 3 |
| Runtime | | | O(n2) |

**5. Explain the advantages and disadvantages of each structure in your evaluation.**

Vectors are simple to implement and use little memory, but printing all courses in order requires sorting at O(n log n) and finding a single course is O(n) unless the vector is maintained in sorted order. Hash tables provide the fastest average lookups at O(1) and make prerequisite validation efficient, but they do not maintain order and require sorting at O(n log n) to print the full course list. Binary search trees naturally keep courses in sorted order, allowing the entire catalog to be printed in O(n) using in-order traversal, and they support searches in O(log n) on average, though worst-case searches degrade to O(n) if the tree is unbalanced. Building the tree can take O(n²) in the worst case scenario but the average would be O(n log n).

**6. Make a recommendation for which data structure you plan to use in your code**.

Based on this analysis, the binary search tree (BST) is the best fit for the advisors’ needs. The main reason is that it allows us to easily print the full list of courses in order, which is something the advisors will use often. With the BST, we can do this in a single pass through the data, rather than having to sort it every time. Looking up an individual course is also very efficient on average, since the BST lets us find the information in just a few steps.

The BST generally works in O(log n) time for searches, which is much faster than a simple list, and it can print all courses in O(n) time. While in the worst case performance could slow down, our dataset is small enough and unlikely to be structured in a way that would cause problems.

Compared to the other options, a vector would either require us to repeatedly sort courses (O(n log n)) or accept slower searches (O(n)). A hash table is excellent at fast lookups, but doesn’t allow us to print courses in order without doing extra sorting. That makes the BST the most practical balance between speed and usability for the advising team.