Pseudocode CS 300 Project 1

Cody Adams

**File Operations:**

// Open file and read data

Open file "course\_data.csv"

While not end of file

Read line from file

Parse line into course fields

Check for formatting errors

End While

Close file

**Course Object Creation:**

// Create course object

Class Course

String courseNumber

String courseName

List<String> prerequisites

Function createCourse(line)

Split line by comma

courseNumber = first field

courseName = second field

prerequisites = remaining fields

Return new Course(courseNumber, courseName, prerequisites)

Print Course Info:

// Print course information and prerequisites

Function printCourseInfo(course)

Print "Course Number: " + course.courseNumber

Print "Course Name: " + course.courseName

If course.prerequisites is not empty

Print "Prerequisites: "

For each prerequisite in course.prerequisites

Print prerequisite

Else

Print "No prerequisites"

**Menu:**

// Menu options

Function displayMenu()

Print "1. Load data into data structure"

Print "2. Print list of courses"

Print "3. Print course information and prerequisites"

Print "9. Exit"

Function menu()

While true

displayMenu()

choice = Read user input

If choice == 1

loadData()

Else If choice == 2

printCourseList()

Else If choice == 3

Print "Enter course number: "

courseNumber = Read user input

printCourseInfo(searchCourse(courseNumber))

Else If choice == 9

Exit

**Sorting and Printing List:**

**/**/ Sort and print courses

Function sortCourses(courses)

Sort courses by courseNumber

Function printCourseList()

sortedCourses = sortCourses(courses)

For each course in sortedCourses

Print course.courseNumber + ": " + course.courseName

**Vector:**

**// Vector - Milestone 1**

Class Course

String courseNumber

String courseName

List<String> prerequisites

Function openFile(String filename)

Open file filename

While not end of file

Read line from file

Parse line into course fields

Check for formatting errors

End While

Close file

Function createCourse(String line) returns Course

Split line by comma

courseNumber = first field

courseName = second field

prerequisites = remaining fields

Return new Course(courseNumber, courseName, prerequisites)

Function loadCourses(Vector<Course> courses, String filename)

Open file filename

While not end of file

line = Read line from file

course = createCourse(line)

courses.add(course)

End While

Close file

Function searchCourse(Vector<Course> courses, String courseNumber) returns Course

For each course in courses

If course.courseNumber == courseNumber

Return course

Return null

Function printCourseInfo(Course course)

Print "Course Number: " + course.courseNumber

Print "Course Name: " + course.courseName

If course.prerequisites is not empty

Print "Prerequisites: "

For each prerequisite in course.prerequisites

Print prerequisite

Else

Print "No prerequisites"

Function sortCourses(Vector<Course> courses)

Sort courses by courseNumber

Function printCourseList(Vector<Course> courses)

sortedCourses = sortCourses(courses)

For each course in sortedCourses

Print course.courseNumber + ": " + course.courseName

Function displayMenu()

Print "1. Load data into data structure"

Print "2. Print list of courses"

Print "3. Print course information and prerequisites"

Print "9. Exit"

Function menu()

Vector<Course> courses = new Vector<Course>()

While true

displayMenu()

choice = Read user input

If choice == 1

Print "Enter filename: "

filename = Read user input

loadCourses(courses, filename)

Else If choice == 2

printCourseList(courses)

Else If choice == 3

Print "Enter course number: "

courseNumber = Read user input

course = searchCourse(courses, courseNumber)

If course is not null

printCourseInfo(course)

Else

Print "Course not found"

Else If choice == 9

Exit

**Evaluation and Analysis:**

Total Cost: *1+1+1+1+1=5*

Runtime: *O(1)*

**Advantages are that it is simple to implement, and can be efficient for smaller datasets.**

**Disadvantages are that it can be slow with larger datasets (O(n).**

**Big O Analysis:**

// Runtime analysis for vector

For all courses: O(n)

If course == courseNumber: O(n)

For each prerequisite: O(n)

Total: O(n^2)

**Hash Table:**

// Hash Table - Milestone 2

Class Course

String courseNumber

String courseName

List<String> prerequisites

Class HashTable

Vector<LinkedList<Course>> table

int tableSize

Function HashTable(int size)

tableSize = size

table = new Vector<LinkedList<Course>>(size)

For i = 0 to size - 1

table[i] = new LinkedList<Course>()

Function hash(String key) returns int

Return key.toInt() % tableSize

Function insert(Course course)

key = hash(course.courseNumber)

table[key].append(course)

Function search(String courseNumber) returns Course

key = hash(courseNumber)

For each course in table[key]

If course.courseNumber == courseNumber

Return course

Return null

Function loadCourses(String filename)

Open file filename

While not end of file

line = Read line from file

course = createCourse(line)

insert(course)

End While

Close file

Function printCourseList()

For each bucket in table

For each course in bucket

Print course.courseNumber + ": " + course.courseName

Function printCourseInfo(Course course)

Print "Course Number: " + course.courseNumber

Print "Course Name: " + course.courseName

If course.prerequisites is not empty

Print "Prerequisites: "

For each prerequisite in course.prerequisites

Print prerequisite

Else

Print "No prerequisites"

Function displayMenu()

Print "1. Load data into data structure"

Print "2. Print list of courses"

Print "3. Print course information and prerequisites"

Print "9. Exit"

Function menu()

HashTable courses = new HashTable(100)

While true

displayMenu()

choice = Read user input

If choice == 1

Print "Enter filename: "

filename = Read user input

courses.loadCourses(filename)

Else If choice == 2

courses.printCourseList()

Else If choice == 3

Print "Enter course number: "

courseNumber = Read user input

course = courses.search(courseNumber)

If course is not null

courses.printCourseInfo(course)

Else

Print "Course not found"

Else If choice == 9

Exit

**Hash Table Analysis:**

Total Cost: *1+n(1+1+1)+1=3n+2*

Runtime: *O(n)*

**Advantages are a faster search, deletion, and insertion. The average O(1)).**

**Some of the disadvantages are collisions, and more memory usage.**

**Big O Analysis:**

// Runtime analysis for hash table

Insert/Search/Delete: O(1) on average

Total: O(n)

**Binary Tree:**

// Binary Search Tree - Milestone 3

Class Course

String courseNumber

String courseName

List<String> prerequisites

Class Node

Course course

Node left

Node right

Class BST

Node root

Function insert(Node node, Course course) returns Node

If node is null

Return new Node(course)

If course.courseNumber < node.course.courseNumber

node.left = insert(node.left, course)

Else

node.right = insert(node.right, course)

Return node

Function remove(Node node, String courseNumber) returns Node

If node is null

Return null

If courseNumber < node.course.courseNumber

node.left = remove(node.left, courseNumber)

Else If courseNumber > node.course.courseNumber

node.right = remove(node.right, courseNumber)

Else

If node.left is null

Return node.right

Else If node.right is null

Return node.left

minNode = findMin(node.right)

node.course = minNode.course

node.right = remove(node.right, minNode.course.courseNumber)

Return node

Function findMin(Node node) returns Node

While node.left is not null

node = node.left

Return node

Function search(Node node, String courseNumber) returns Course

If node is null or node.course.courseNumber == courseNumber

Return node.course

If courseNumber < node.course.courseNumber

Return search(node.left, courseNumber)

Else

Return search(node.right, courseNumber)

Function loadCourses(String filename)

Open file filename

While not end of file

line = Read line from file

course = createCourse(line)

root = insert(root, course)

End While

Close file

Function inOrder(Node node, Vector<Course> sortedCourses)

If node is not null

inOrder(node.left, sortedCourses)

sortedCourses.add(node.course)

inOrder(node.right, sortedCourses)

Function printCourseList()

Vector<Course> sortedCourses

inOrder(root, sortedCourses)

For each course in sortedCourses

Print course.courseNumber + ": " + course.courseName

Function printCourseList()

inOrder(root)

Function printCourseInfo(Course course)

Print "Course Number: " + course.courseNumber

Print "Course Name: " + course.courseName

If course.prerequisites is not empty

Print "Prerequisites: "

For each prerequisite in course.prerequisites

Print prerequisite

Else

Print "No prerequisites"

Function displayMenu()

Print "1. Load data into data structure"

Print "2. Print list of courses"

Print "3. Print course information and prerequisites"

Print "9. Exit"

Function menu()

BST courses = new BST()

While true

displayMenu()

choice = Read user input

If choice == 1

Print "Enter filename: "

filename = Read user input

courses.loadCourses(filename)

Else If choice == 2

courses.printCourseList()

Else If choice == 3

Print "Enter course number: "

courseNumber = Read user input

course = courses.search(courses.root, courseNumber)

If course is not null

courses.printCourseInfo(course)

Else

Print "Course not found"

Else If choice == 9

Exit

**Tree Analysis:**

Total Cost: *1+logn+logn+1=2logn+2*

Some advantages are that it is efficient at search, deletion, and insertion. (O(log n) for balanced trees).

Some disadvantages are they are complicated to implement, and have more overhead.

**Big O Analysis:**

// Runtime analysis for tree

Insert/Search/Delete: O(log n)

Total: O(log n)

**Recommendation:**

**Using the hash table for its overall average case efficacy. This provides great performance for insertion, search, and deletion operations, which is great because it is important for us to manage course information efficiently.**