Ben Douglas

06-12-2024

6-2 Submit Project One- Assignment

***Pseudocode and Runtime Analysis for Course Advising Program***

Vector Data Structure

Pseudocode:

1.)Open and Read File:

function readFile(fileName):

open file fileName for reading

for each line in the file:

parse line into course number, course title, and prerequisites

if line is improperly formatted:

print error message

continue to next line

create a course object with parsed data

add course object to vector

close file

2.)Create Course Object:

function createCourseObject(line):

split line by comma

set course number to first element

set course title to second element

set prerequisites to remaining elements

return new Course object

3.)Print Course Information:

function printCourseInformation(courses, courseNumber):

for each course in courses:

if course number matches courseNumber:

print course title

print prerequisites

4.)Menu Options:

function menu():

print "1. Load course data"

print "2. Print all courses in alphanumeric order"

print "3. Print course information"

print "9. Exit"

choice = get user input

if choice == 1:

fileName = get user input for file name

readFile(fileName)

else if choice == 2:

sortAndPrintCourses(courses)

else if choice == 3:

courseNumber = get user input for course number

printCourseInformation(courses, courseNumber)

else if choice == 9:

exit program

5.)Sort and Print Courses:

function sortAndPrintCourses(courses):

sort courses by course number

for each course in courses:

print course number and title

***Runtime Analysis:***

Reading File:

-Line Cost: 1 (constant time for parsing each line)

-Times Executes: n (number of lines/courses)

-Total Cost: O(n)

Creating Course Objects:

-Line Cost: 1 (creating each course object)

-Times Executes: n

-Total Cost: O(n)

Sorting Courses:

-Line Cost: 1 (comparison during sorting)

-Times Executes: n log n (typical for sorting algorithms like quicksort or mergesort)

-Total Cost: O(n log n)

Searching for a Course:

-Line Cost: 1 (checking each course)

-Times Executes: n

-Total Cost: O(n)

------------------------------------------------------------------

*Hash Table Data Structure*

Pseudocode:

1.)Open and Read File:

function readFile(fileName):

open file fileName for reading

for each line in the file:

parse line into course number, course title, and prerequisites

if line is improperly formatted:

print error message

continue to next line

create a course object with parsed data

add course object to hash table

close file

2.)Create Course Object:

function createCourseObject(line):

split line by comma

set course number to first element

set course title to second element

set prerequisites to remaining elements

return new Course object

3.)Print Course Information:

function printCourseInformation(courses, courseNumber):

if courseNumber exists in hash table:

print course title

print prerequisites

else:

print "Course not found"

4.)Menu Options:

function menu():

print "1. Load course data"

print "2. Print all courses in alphanumeric order"

print "3. Print course information"

print "9. Exit"

choice = get user input

if choice == 1:

fileName = get user input for file name

readFile(fileName)

else if choice == 2:

sortAndPrintCourses(courses)

else if choice == 3:

courseNumber = get user input for course number

printCourseInformation(courses, courseNumber)

else if choice == 9:

exit program

5.)Sort and Print Courses:

function sortAndPrintCourses(courses):

extract all courses from hash table

sort courses by course number

for each course in sorted list:

print course number and title

***Runtime Analysis:***

Reading File:

-Line Cost: 1

-Times Executes: n

-Total Cost: O(n)

Creating Course Objects:

-Line Cost: 1

-Times Executes: n

-Total Cost: O(n)

Inserting into Hash Table:

-Line Cost: 1 (amortized, assuming good hash function)

-Times Executes: n

-Total Cost: O(n)

Searching for a Course:

-Line Cost: 1 (amortized)

-Times Executes: 1 (constant time lookup)

-Total Cost: O(1)

Sorting Courses:

-Line Cost: 1

-Times Executes: n log n

-Total Cost: O(n log n)

---------------------------------------------------------------------

*Binary Search Tree (BST) Data Structure*

Pseudocode:

1.)Open and Read File:

function readFile(fileName):

open file fileName for reading

for each line in the file:

parse line into course number, course title, and prerequisites

if line is improperly formatted:

print error message

continue to next line

create a course object with parsed data

add course object to BST

close file

2.)Create Course Object:

function createCourseObject(line):

split line by comma

set course number to first element

set course title to second element

set prerequisites to remaining elements

return new Course object

3.)Print Course Information:function printCourseInformation(courses, courseNumber):

search BST for courseNumber

if found:

print course title

print prerequisites

else:

print "Course not found"

4.)Menu Options:

function menu():

print "1. Load course data"

print "2. Print all courses in alphanumeric order"

print "3. Print course information"

print "9. Exit"

choice = get user input

if choice == 1:

fileName = get user input for file name

readFile(fileName)

else if choice == 2:

inOrderTraversalAndPrint(courses)

else if choice == 3:

courseNumber = get user input for course number

printCourseInformation(courses, courseNumber)

else if choice == 9:

exit program

5.)In-Order Traversal and Print:

function inOrderTraversalAndPrint(node):

if node is not null:

inOrderTraversalAndPrint(node.left)

print node.course number and title

inOrderTraversalAndPrint(node.right)

***Runtime Analysis:***

-Reading File:

-Line Cost: 1

-Times Executes: n

-Total Cost: O(n)

Creating Course Objects:

-Line Cost: 1

-Times Executes: n

-Total Cost: O(n)

Inserting into BST:

-Line Cost: log n (on average)

-Times Executes: n

-Total Cost: O(n log n)

Searching for a Course:

-Line Cost: log n (on average)

-Times Executes: 1

-Total Cost: O(log n)

In-Order Traversal:

-Line Cost: 1

-Times Executes: n

-Total Cost: O(n)

-----------------------------------------------------------------

***Evaluation and Recommendation***

Vector:

Advantages: Simple to implement and understand, good for small datasets.

Disadvantages: Searching and insertion are O(n) in the worst case, which can be inefficient for large datasets.

Hash Table:

Advantages: Average case for insertion and search operations is O(1), making it very efficient.

Disadvantages: Worst-case time complexity for insertion and search can be O(n) if there are many collisions. Memory overhead can be higher due to space allocated for the hash table.

Binary Search Tree (BST):

Advantages: Keeps elements sorted, allowing for efficient in-order traversal. Average case time complexity for insertion and search is O(log n).

Disadvantages: In the worst case (unbalanced tree), time complexity for insertion and search can degrade to O(n).

Recommendation:

Based on the analysis, the Hash Table data structure is recommended for this application due to its average case O(1) time complexity for insertion and search operations. This makes it highly efficient for the expected operations. Although the worst-case scenario is O(n), this is rare with a good hash function, and the benefits in the average case outweigh this potential downside.