Kelly Perez

06/05/2023

CS 300

Project One

**Pseudocode**

struct Course

string number

string title

vector<string> prerequisites

Course(courseNum, courseTitle)

number = courseNum

title = courseTitle

addPrerequisite(prerequisite)

prerequisites.push\_back(prerequisite)

getNumber()

return number

getTitle()

return title

getPrerequisites()

return prerequisites

function createHashTable()

// Code to create and return a hash table data structure

hashTable = new HashTable()

return hashTable

function createVector()

// Code to create and return a vector data structure

vector = new Vector()

return vector

function createBinarySearchTree()

// Code to create and return a binary search tree data structure

binarySearchTree = new BinarySearchTree()

return binarySearchTree

void parseFile(filename, dataStructure)

file = openFile(filename)

if fileExists(filename) is false

displayErrorMessage("Failed to open the file")

while endOfFile(file) is false

line = readLine(file)

tokens = splitLine(line, ' ')

if length(tokens) < 2

displayErrorMessage("Invalid file format")

continue

courseNum = tokens[0]

courseTitle = tokens[1]

prerequisites = tokens[2:]

course = new Course(courseNum, courseTitle)

for each prerequisite in prerequisites

course.addPrerequisite(prerequisite)

dataStructure.insert(course)

closeFile(file)

function printCourseInformation(dataStructure, courseNum)

course = dataStructure.find(courseNum)

if course is null

displayErrorMessage("Course not found")

else

print(course.getNumber(), course.getTitle())

for each prerequisite in course.getPrerequisites()

prerequisiteCourse = dataStructure.find(prerequisite)

if prerequisiteCourse is not null

print(prerequisiteCourse.getNumber(), prerequisiteCourse.getTitle())

function printAlphanumericCourseList(dataStructure)

courseList = dataStructure.getSortedCourseList()

for each course in courseList

print(course.getNumber(), course.getTitle())

function main()

filename = "Courses.txt"

dataStructure = createHashTable()

parseFile(filename, dataStructure)

while true

displayMenu()

choice = getUserChoice()

if choice is "1"

printAlphanumericCourseList(dataStructure)

else if choice is "2"

courseNum = getCourseNumberFromUser()

printCourseInformation(dataStructure, courseNum)

else if choice is "3"

exitProgram()

else

displayErrorMessage("Invalid choice")

**Evaluation**

**Vector**:

Advantages: Easy installation.

Indexes elements efficiently and randomly.

Dynamic resizing offers flexible storage.

Disadvantages:

Inserting and deleting repeatedly in the vector can be inefficient.

Large vectors may need expensive memory reallocation and element copying when resizing.

Searching a vector needs sequential iteration.

**Hash Table**:

Advantages:

Stores and retrieves key-value pairs efficiently (O(1) average).

Ideal for fast element access.

Chains or addresses collisions.

Disadvantages:

Needs a good hash function to distribute elements uniformly and reduce collisions.

Collisions require more memory.

Hash table iteration may change the insertion order.

**Binary Search Tree (BST):**

Advantages:

Sorts elements for efficient searching, insertion, and deletion (O(log n) average case).

In-order traversal allows element sorting.

For data sorting.

Disadvantages:

Tree balance affects performance. Unbalanced trees can reduce operations to O(n).

Needs memory for left and right child node references.

Inefficient for dynamic resizing and restructuring.

**Recommendation**:

The software should use a hash table, according to the analysis. Printing course information and prerequisites requires quick look up and retrieval procedures. It handles enormous amounts of data better than a vector or binary search tree.