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 sortVector(vector)

sortedVector = copy(vector)

sort(sortedVector) // Perform sorting operation on the sortedVector

return sortedVector

function sortHashTable(hashTable)

courseList = []

for each bucket in hashTable

for each course in bucket

courseList.push(course)

sortedList = sort(courseList) // Perform sorting operation on the courseList

return sortedList

function inOrderTraversal(node, sortedList)

if node is null

return

inOrderTraversal(node.left, sortedList)

sortedList.push(node.course)

inOrderTraversal(node.right, sortedList)

function sortBinarySearchTree(binarySearchTree)

sortedList = createVector()

inOrderTraversal(binarySearchTree.root, sortedList)

sortedList = sortVector(sortedList) // Perform sorting operation on the sortedList

return sortedList

function printSortedCourseList(dataStructure)

sortedList = dataStructure.getSortedCourseList()

for each course in sortedList

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

function main()

filename = "Courses.txt"

dataStructure = createHashTable()

vector = createVector()

binarySearchTree = createBinarySearchTree()

parseFile(filename, dataStructure)

parseFile(filename, vector)

parseFile(filename, binarySearchTree)

while true

displayMenu()

choice = getUserChoice()

if choice is "1"

sortedVector = sortVector(vector)

printSortedCourseList(sortedVector)

else if choice is "2"

sortedHashTable = sortHashTable(hashTable)

printSortedCourseList(sortedHashTable)

else if choice is "3"

sortedBinarySearchTree = sortBinarySearchTree(binarySearchTree)

printSortedCourseList(sortedBinarySearchTree)

else if choice is "4"

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.