Project One Pseudocode & Runtime Analysis

**Vector Data Structure**

Struct Course:

courseNumber

name

prerequisites (vector of strings)

Function readFile(filePath):

Open file at filePath

If file is not found:

Print “Error. File not found.”

return empty vector

else:

create an empty vector named lines

while not end of file:

read line

if number of values in line < 2:

print “Error. Invalid line format.”

continue

add line to lines vector

close file

return lines

Function createCourseObjects(lines):

create an empty vector named courses

create an empty dictionary named courseMap

for each line in lines:

split line by commas into tokens

courseNumber = tokens[0]

name = tokens[1]

prerequisites = vector of tokens from index 2 to end

if courseNumber is empty or name is empty:

print "Error. Invalid course number or name."

continue

for prerequisite in prerequisites:

if prerequisite is not in courseMap:

print "Error: Prerequisite " + prerequisite + " not found"

continue

create a Course object with courseNumber, name, and prerequisites

add the Course object to courses vector

add courseNumber to courseMap with value true

return courses

Function searchCourse(courses, courseNumber):

for course in courses:

if course.courseNumber == courseNumber:

print "Course Number: " + course.courseNumber

print "Course Name: " + course.name

print "Prerequisites: "

for prerequisite in course.prerequisites:

print " - " + prerequisite

return

print "Course " + courseNumber + " not found."

Function sortCourses(courses):

for i from 0 to length(courses) - 1:

for j from 0 to length(courses) - 1 - i:

if courses[j].courseNumber > courses[j + 1].courseNumber:

swap courses[j] and courses[j + 1]

return courses

Function printSortedCourses(courses):

sortedCourses = sortCourses(courses)

for course in sortedCourses:

print "Course Number: " + course.courseNumber

print "Course Name: " + course.name

Function displayMenu():

print "1. Load course data from file"

print "2. Print all courses in alphanumeric order"

print "3. Print course title and prerequisites"

print "9. Exit"

Function main():

courses = vector()

while True:

displayMenu()

choice = input("Enter your choice: ")

if choice == "1":

filePath = input("Enter file path: ")

lines = readFile(filePath)

courses = createCourseObjects(lines)

elif choice == "2":

printSortedCourses(courses)

elif choice == "3":

courseNumber = input("Enter course number: ")

searchCourse(courses, courseNumber)

elif choice == "9":

break

else:

print "Invalid choice. Please try again."

**Hash Table Data Structure**

Struct Course:

courseNumber

name

prerequisites (vector of strings)

Struct Node:

course (Course object)

next (pointer to next Node)

Struct HashTable:

vector<Node> table

int size

Function hash(courseNumber):

return hash code for courseNumber % table.size

Function insert(hashTable, course):

key = hash(course.courseNumber)

if hashTable.table[key] is empty:

hashTable.table[key] = Node(course, null)

else:

current = hashTable.table[key]

while current.next is not null:

current = current.next

current.next = Node(course, null)

Function openFile(fileName):

Open file with fileName

If file is not found:

Print "Error, file not found."

return null

else:

return file

Function readFile(file, hashTable):

while not end of file:

read line

split line by commas into tokens

courseNumber = tokens[0]

name = tokens[1]

prerequisites = vector of tokens from index 2 to end

course = Course(courseNumber, name, prerequisites)

insert(hashTable, course)

Function searchKey(hashTable, courseNumber):

key = hash(courseNumber)

current = hashTable.table[key]

while current is not null:

if current.course.courseNumber == courseNumber:

return current.course

current = current.next

return null

Function extractCourses(hashTable):

courses = vector()

for each node in hashTable.table:

current = node

while current is not null:

add current.course to courses

current = current.next

return courses

Function sortCourses(courses):

for i from 0 to length(courses) - 1:

for j from 0 to length(courses) - 1 - i:

if courses[j].courseNumber > courses[j + 1].courseNumber:

swap courses[j] and courses[j + 1]

return courses

Function printSortedCourses(hashTable):

courses = extractCourses(hashTable)

sortedCourses = sortCourses(courses)

for course in sortedCourses:

print "Course Number: " + course.courseNumber

print "Course Name: " + course.name

Function printCourseInfo(course):

print "Course Number: " + course.courseNumber

print "Course Name: " + course.name

print "Prerequisites: "

for prerequisite in course.prerequisites:

print " - " + prerequisite

Function displayMenu():

print "1. Load course data from file"

print "2. Print all courses in alphanumeric order"

print "3. Print course title and prerequisites"

print "9. Exit"

Function main():

hashTable = HashTable(size=10)

while True:

displayMenu()

choice = input("Enter your choice: ")

if choice == "1":

filePath = input("Enter file path: ")

file = openFile(filePath)

if file is not null:

readFile(file, hashTable)

elif choice == "2":

printSortedCourses(hashTable)

elif choice == "3":

courseNumber = input("Enter course number: ")

course = searchKey(hashTable, courseNumber)

if course is not null:

printCourseInfo(course)

else:

print "Course not found."

elif choice == "9":

break

else:

print "Invalid choice. Please try again."

**Tree Data Structure**

Struct Course:

courseNumber

name

prerequisites (vector of strings)

Struct TreeNode:

course (Course object)

left (pointer to left TreeNode)

right (pointer to right TreeNode)

Struct Tree:

root (TreeNode)

Function insertCourse(node, course):

if node is null:

return TreeNode(course, null, null)

if course.courseNumber < node.course.courseNumber:

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

else if course.courseNumber > node.course.courseNumber:

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

return node

Function openFile(fileName):

Open file with fileName

If file is not found:

Print "Error. File not found."

return null

else:

return file

Function readFile(file, tree):

while not end of file:

read line

split line by commas into tokens

courseNumber = tokens[0]

name = tokens[1]

prerequisites = vector of tokens from index 2 to end

course = Course(courseNumber, name, prerequisites)

tree.root = insertCourse(tree.root, course)

Function searchTree(node, courseNumber):

if node is null or node.course.courseNumber == courseNumber:

return node

if courseNumber < node.course.courseNumber:

return searchTree(node.left, courseNumber)

else:

return searchTree(node.right, courseNumber)

Function inOrderTraversal(node, courses):

if node is not null:

inOrderTraversal(node.left, courses)

add node.course to courses

inOrderTraversal(node.right, courses)

Function printSortedCourses(tree):

courses = vector()

inOrderTraversal(tree.root, courses)

for course in courses:

print "Course Number: " + course.courseNumber

print "Course Name: " + course.name

Function printCourseInfo(course):

print "Course Number: " + course.courseNumber

print "Course Name: " + course.name

print "Prerequisites: "

for prerequisite in course.prerequisites:

print " - " + prerequisite

Function displayMenu():

print "1. Load course data from file"

print "2. Print all courses in alphanumeric order"

print "3. Print course title and prerequisites"

print "9. Exit"

Function main():

tree = Tree()

while True:

displayMenu()

choice = input("Enter your choice: ")

if choice == "1":

filePath = input("Enter file path: ")

file = openFile(filePath)

if file is not null:

readFile(file, tree)

elif choice == "2":

printSortedCourses(tree)

elif choice == "3":

courseNumber = input("Enter course number: ")

node = search

**Vector Data Structure:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| Open file at ‘filePath’ | 1 | 1 | 1 |
| Check if file is not found | 1 | 1 | 1 |
| While not end of file | 1 | n | n |
| Read line | 1 | n | n |
| Split line by commas | 1 | n | n |
| Validate format and prerequisites | 1 | n | n |
| Create ‘Course’ object | 1 | n | n |
| Add ‘Course’ to vector | 1 | n | n |
| **Total Cost** | | | 6n+2 |
| **Runtime** | | | O(n) |

**Hash Table Data Structure:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| Open file | 1 | 1 | 1 |
| Check if file is not found | 1 | 1 | 1 |
| While not end of file | 1 | n | n |
| Read line | 1 | n | n |
| Split line into parameters | 1 | n | n |
| Validate format and prerequisites | 1 | n | n |
| Create ‘Course’ object | 1 | n | n |
| Calculate hash key | 1 | n | n |
| Insert into hash table | 1 | n | n |
| **Total Cost** | | | 7n+2 |
| **Runtime** | | | O(n) |

**Tree Data Structure:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| Open file | 1 | 1 | 1 |
| Check if file is not found | 1 | 1 | 1 |
| While not end of file | 1 | n | n |
| Read line | 1 | n | n |
| Split line into tokens | 1 | n | n |
| Validate format and prerequisites | 1 | n | n |
| Create ‘Course’ object | 1 | n | n |
| Insert into tree | 1 | n | n |
| **Total Cost** | | | 6n+2 |
| **Runtime** | | | O(n) |

When choosing between vectors, hash tables, and trees, each has its advantages and disadvantages. Vectors are simple and offer fast access by index, making them easy to implement and great for tasks where data doesn’t change often. However, they struggle with frequent insertions or deletions and don’t maintain an inherent order, so it needs to be sorted manually. On the other hand, hash tables are fantastic for fast lookups, additions, and deletions, making them ideal for large datasets. They do require more memory and can be complex due to potential collisions, and since they don’t maintain order, sorting the data can be cumbersome. Trees provide a balanced approach, efficiently managing sorted data and offering fast operations like insertions, deletions, and lookups. However, they’re more complex to implement and can slow down if not properly balanced.

Based on the runtime analysis, vectors are the best choice for this project. Their simplicity and efficiency in reading files and adding course objects, with a runtime of 6n + 2, make them the fastest and most practical option. Vectors offer the straightforward implementation and quick access needed. While they might not be ideal for more dynamic data, their performance and ease of use make them the most suitable option for the specific requirements of this project.

**Citations**

Zybooks. (n.d.). https://learn.zybooks.com/zybook/CS-300-11203.202456-1/chapter/3/section/7

Karimov, E. (2024, May 25). *Understanding hash tables: Implementation and best practices*. Understanding Hash Tables: Implementation and Best Practices. <https://elshadk.substack.com/p/understanding-hash-tables-implementation>

GeeksforGeeks. (2023, March 23). *Applications, advantages and disadvantages of Tree*. <https://www.geeksforgeeks.org/applications-advantages-and-disadvantages-of-tree/>

Usman, M. (2024, May 16). *Vector Data Structure*. CodeLikeChamp. https://codelikechamp.com/vector-data-structure/