CS-300

Mohamed. Elmarzougui

Southern New Hampshire University

6-2 Project One

**Pseudocode for Vector Data Structure:**

// Define Course class

class Course {

int courseNumber

string courseTitle

List<string> prerequisites

}

// Function to load data from file into a vector data structure

function loadDataIntoVector(fileName, courseVector):

fileContents = readFile(fileName)

if fileContents is empty:

return

courseList = createCourseList(fileContents)

for each course in courseList:

courseVector.push\_back(course)

// Function to print course information and prerequisites from vector data structure

function printCourseInfoFromVector(courseVector, targetCourseNumber):

targetCourse = findCourse(courseVector, targetCourseNumber)

printCourseInfo(targetCourse, targetCourseNumber)

// Function to print a sorted list of courses from vector data structure

function printSortedCourseListFromVector(courseVector):

sortedCourseVector = sortCourses(courseVector)

for each course in sortedCourseVector:

printCourseInfo(course, course.courseNumber)

**Pseudocode for Hash Table Data Structure:**

// Define Course class

class Course {

int courseNumber

string courseTitle

List<string> prerequisites

}

// Define HashTable class

class HashTable {

vector<Node> nodes

// Method to insert course into hash table

method Insert(course):

key = hash(course.courseNumber)

nodes[key].insert(course)

// Method to search for course in hash table

method Search(key):

// Implementation of search method omitted for brevity

}

// Function to load data from file into a hash table data structure

function loadDataIntoHashTable(fileName, hashTable):

fileContents = readFile(fileName)

if fileContents is empty:

return

courseList = createCourseList(fileContents)

for each course in courseList:

hashTable.Insert(course)

// Function to print course information and prerequisites from hash table data structure

function printCourseInfoFromHashTable(hashTable, targetCourseNumber):

course = hashTable.Search(targetCourseNumber)

printCourseInfo(course, targetCourseNumber)

**Pseudocode for Tree Data Structure:**

Copy code

// Define Course class

class Course {

int courseNumber

string courseTitle

List<string> prerequisites

}

// Define TreeNode class

class TreeNode {

Course course

List<TreeNode> children

}

// Function to load data from file into a tree data structure

function loadDataIntoTree(fileName):

fileContents = readFile(fileName)

if fileContents is empty:

return null

courseList = createCourseList(fileContents)

// Implementation of creating tree structure from course list omitted for brevity

return root // Return root of the tree

// Function to print course information and prerequisites from tree data structure

function printCourseInfoFromTree(root, targetCourseNumber):

targetCourse = findCourse(root, targetCourseNumber)

printCourseInfo(targetCourse, targetCourseNumber)

// Function to print a sorted list of courses from tree data structure

function printSortedCourseListFromTree(root):

// Implementation of printing sorted list from tree structure omitted for brevity

**Menu Pseudocode:**

// Pseudocode for menu

function menu():

while true:

print("1. Load Data Structure")

print("2. Print Course List")

print("3. Print Course")

print("4. Exit")

choice = getUserInput()

if choice == 1:

// Call function to load data into chosen data structure

else if choice == 2:

// Call function to print course list

else if choice == 3:

// Call function to print course information

else if choice == 4:

break

else:

print("Invalid choice. Please select again.")

**Runtime Analysis:**

* Vector Data Structure:

Loading Data: O(n)

Printing Course Information: O(n)

Printing Sorted Course List: O(n log n) - Sorting cost

* Hash Table Data Structure:

Loading Data: O(n)

Printing Course Information: O(1) (Average case)

Printing Sorted Course List: Not applicable (Hash tables don't maintain order)

* Tree Data Structure:

Loading Data: O(n log n) - Assuming balanced tree

Printing Course Information: O(log n) - Assuming balanced tree

Printing Sorted Course List: O(n log n) - Inorder traversal of balanced tree

**Advantages and Disadvantages:**

* Vector:

Advantages: Simple to implement, supports efficient random access.

Disadvantages: Insertion and deletion may be slow, especially if resizing is required.

* Hash Table:

Advantages: Offers constant time complexity for search (on average), suitable for large datasets.

Disadvantages: Does not maintain order, collisions can occur which may degrade performance.

* Tree:

Advantages: Maintains order, efficient for both search and printing in sorted order.

Disadvantages: Insertion and deletion may be slower compared to hash table, and may require balancing for optimal performance.

**Recommendation:**

I recommend using the Hash Table data structure. Although it doesn't maintain order inherently, but it offers efficient search (O(1) on average) for printing course information, which is a common operation in the application and it provides the necessary functionality and performance for this specific task.