**Module 6-2 Project One**

Mitchel Harmon

Online Campus, Southern New Hampshire University

CS-300-H2997 DSA: Analysis and Design

Instructor Nathan Lebel

December 2, 2023

**Module 6-2 Project One**

**Opening and Reading File**

file\_path = file location

Try:

file = open file\_path

Except file not found:

Print file not found

Intalize courses\_vector,hash table,or binary tree

For each line in file

Create stringstream with line contents

While stringstream has more tokens separated by comma

Course\_data = get next stringstream token

If valid\_format(course\_data)

course\_object = create\_course(course\_data)

// for vector append courses\_vector with course\_object//

//for hash table hashTable.insert(courseNumber, course\_object)//

//for binary tree

Tree.addNode(course\_object)

//

Close file

**Function for creating course objects**

Function create\_course(course\_data)

course\_num = course\_data[0]

course\_title = course\_data[1]

prereqs = course\_data[2:]

course\_object = course(course\_num, course\_title, prereqs)

return course object

**Function for validating format**

Function validate\_format(course\_data)

if the length of course\_data < 2

print error

return false

else

return true

**Pseudocode for adding node(Binary Tree)**

AddNode(Course course)

TreeNode\* newNode = new TreeNode

NewNode-> = course

Nodes[course number] = newNode

**Pseudocode for Searching and Printing Course Information(vector)**

Function print\_course(course\_num)

Intalize searched\_course = null

For course in courses\_vector

if course in course\_vector equals course\_num

seatched\_course = course

break

If course was found

print course info

if course has prerequisites

print prereqs

Else

print no course found

**Pseudocode for Searching and Printing Course Information(Hash Table)**

Function print\_course(hashTable)

for key in hashTable.keys():

course = hashTable.get(key)

print course number

print title

if len(course.prerequisites) > 0:

Print prerequisites

**Pseudocode for Searching and Printing Course Information(Binary Tree)**

Function print\_course(TreeNode courseNode)

If root in not null

Output course number // courseNode->course number

Output course title // courseNode->course title

If courseNode->course prereqs is not empty

For prereqs node : couseNode->children

Output course prereqs

**Pseudocode for menu**

Menu()

print menu options

get valid choice input

if choice is 1

load courses from file

if choice is 2

print courses

if choice is 3

request course number

print information based on course number

if choice is 4

exit program

else

invalid input

**Sorting for vector**

function sortCoursesByNumber(courses\_vector)

for i from 1 TO courses\_vector.length - 1

key = courses\_vector[i]

j = i - 1

while j >= 0 and courses\_vector[j] > key

Courses\_vector[j + 1] = courses[j]

j = j - 1

courses[j + 1] = key

return courses

function printSortedCourses(courses)

sortedCourses = sortCoursesByNumber(courses\_vector)

for each course in sortedCourses

print course

**Sorting for hash table**

function sortKeys(keys)

for i from 1 TO keys.length - 1

key = keys[i]

j = i - 1

while j >= 0 and keys[j] > key

keys[j + 1] = keys[j]

j = j - 1

keys[j + 1] = key

return keys

FUNCTION printSortedCourses(hashTable)

keys = hashTable.keys()

sortedKeys = sortKeys(keys)

for each key in sortedKeys

print hashTable[key]

**Sorting for binary tree**

function insertIntoTree(binaryTree, course)

if binaryTree is empty

create new node with course

set binaryTree root as new node

else

if course number of binaryTree root > course number of course

set binaryTree left child as insertIntoTree(binaryTree left child, course)

else

set binaryTree right child as insertIntoTree(binaryTree right child, course)

return binaryTree

function printInOrderTraversal(binaryTree)

if binaryTree is not empty

inOrderTraversal(binaryTree left child)

print binaryTree course information

inOrderTraversal(binaryTree right child)

**Runtime Analysis**

|  |  |  |  |
| --- | --- | --- | --- |
| **Vector** | **Line Cost** | **# Times executes** | **Total Cost** |
| Create vector | 1 | 1 | 1 |
| For each line in file | 1 | n | n |
| Is Valid | 1 | 1 | 1 |
| If course\_data.length < 2 | 1 | n | n |
| Create course object | 1 | n | n |
| prereq | 1 | n | n |
| Append course object | 1 | n | n |
| Total | | | 6n+1 |
| Runtime | | | O(n) |

|  |  |  |  |
| --- | --- | --- | --- |
| **Hash Table** | **Line cost** | **# times executes** | **Total Cost** |
| Create hash table | 1 | 1 | 1 |
| For each line in file | 1 | n | n |
| Is valid | 1 | n | n |
| If course\_data.length < 2 | 1 | n | n |
| Create course object | 1 | n | n |
| prereq | 1 | n | n |
| Insert into hash table | 1 | n | n |
| Total | | | 6n+1 |
| Runtime | | | O(n) |

|  |  |  |  |
| --- | --- | --- | --- |
| **Binary Tree** | **Line Cost** | **# times executes** | **Total Cost** |
| Create tree | 1 | 1 | 1 |
| For each line in file | 1 | n | n |
| Is valid | 1 | n | n |
| If course\_data.length < 2 | 1 | n | n |
| Create course object | 1 | n | n |
| prereq | 1 | n | n |
| Tree.addnode | 1 | n | n |
| Create new node | 1 | n | n |
| NewNode-> = course | 1 | n | n |
| Nodes[course number] = newNode | 1 | n | n |
| Total | | | 9n+1 |
| Runtime | | | O(n) |

**Advantages and disadvantages of the data structures**  
 The advantages of using a vector is that is relatively simple to implement do to it allowing sequential access and maintaining the order of elements. The disadvantages of using vector are that searching the vector can take a lot of time as it must search each element individually to find the correct element.

The advantages of hash tables are that it can search through the list quickly through the use of keys. The disadvantages of using a hash table are that it might suffer from collisions which degrade the performance also it does not maintain order.

The advantages of a binary tree are that it maintains sorted order of elements and is efficient for searching elements. The disadvantages of using a binary tree are that the insertion and deletion operations are typically more complex than using a hash table or vector.

**Recommendation for data structure**

Given the requirements for sorting and efficient retrieval, using a self-balancing binary search tree seems to be a suitable choice. Although the initial reading of the file might take longer due to tree insertion, subsequent retrieval and sorting operations will be more efficient.