Alexis Prazak

12/12/2024

Michael Susalla

CS300 Project One

**Read Data**

open file with fstream

void loadCourses(csv file, data structure)

if file does not open

print error

else check each file line

if less ten two parameters

print missing information error

void matchPrerequisite(Vector<Course> courses, String courseNumber){

if course does not have prerequisite

return

else

call searchCourse for that prerequisite }

**Vector Data Structure**

Vector<Course>

create course object struct

set vector size to lines in the file

while i is less then or equal to size

add current line to vector

**Course Search**

**//Vector**

void searchCourse(Vector<Course> courses, String courseNumber) {

declare current variable and set equal to head node

for all courses

if the course is the same as courseNumber

print out the course information

for each prerequisite of the course

print the prerequisite course information

if course does not match courseNumber

check next node

if current node is tail node

print that course does not exist

}

**//Hash Table**

define HashTable<Course>

define node structure

define tableSize

declare int key

void searchCourse(HashTable<Course> courses, String courseNumber) {

define key node

if key node found and empty

assign node to key

if key node NOT found

check next node

else if node contains course

Assign to next empty node

}

void loadCourse (HashTable<Course> courses, String courseNumber) {

for each row in file

set courseId

set courseName

if there is a prerequisite

set prerequisite

searchCourse()

}

**//Binary Tree**

define BinaryTree<Course>

void searchCourse(Tree<Course> courses, String courseNumber)

set current node

while node is not null

if course id matches current

output course information

output prerequisite information(if it exists)

else if course id is lower than current

Set current to left node

else

set current to right node

**//Menu**

int main

Set choice equal to 0

cout << Menu

<< 1:Load Courses << 2:Print All Courses << 3:Print Selected Course << 9:Exit Program

while choice is not 9:

case 1

loadCourses(csv file, data structure)

case 2

printSorted

case 3

int course

cin >> course

printCourse(course)

case 9

return

return

**//Print Courses**

displayCourses(parameter for sorted vector)

for each element in sorted vector

cout << courseID << courseName << course prerequisites

**//Vector**

printVector(sorted vector)

declare sorted<Course> and set size to list size

//Partition

lowIndex equals first course number

highIndex equals last course number

pivot equals lowIndex + (highIndex - lowIndex) / 2

done is False

while done is False

while lowIndex is less then pivot

increment lowIndex by 1

while pivot is less than highIndex

decrement highIndex by 1

if lowIndex is greater than highIndex

done is True

else

swap lowIndex and highIndex

return highIndex

quickSort(Vector, lowIndex, highIndex)

if lowIndex is greater than highIndex

return

recursively call quickSort

call displayCourses

**//Hash Table**

//since hash tables are already sorted the existing data structure can be printed

call displayCourses(HashTable<Course>)

**//Binary Tree**

void inOrder(node)

declare sorted vector

if node is not null

inOrder(node->left)

add node to sorted vector

inOrder(node->right)

add node to sorted vector

call displayCourses(sorted vector)

## **Runtime Analysis**

| **Vector** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **Create vector** | 1 | 1 | 1 |
| **Set vector size to lines in file** | 1 | n | n |
| **While i is less than or equal to size** | 1 | n | n |
| **Add current line to vector** | 1 | n | n |
| **Return vector** | 1 | 1 | 1 |
| **Total Cost** | | | 3n+2 |
| **Runtime** | | | O(N) |

| **Hash Table** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **Create hash table** | 1 | 1 | 1 |
| **Define key node** | 1 | 1 | 1 |
| **Check if key node is null** | 1 | n | n |
| **Assign node to key** | 1 | n | n |
| **If key node not found, Check next node** | 2 | n | 2n |
| **Check if node contains course** | 1 | n | n |
| **Assign to next empty node** | 1 | n | n |
| **loadCourses method** | 0 | 0 | 0 |
| **For each line in file** | 1 | n | n |
| **Set course details & prereq details(if prereq exists)** | 3 | n | 3n |
| **Insert course** | 1 | n | n |
| **Total Cost** | | | 11n+2 |
| **Runtime** | | | O(N) |

| **Binary Tree** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **Create node structure** | 0 | 0 | 0 |
| **Set current node to root** | 1 | 1 | 1 |
| **Check if root node is not null** | 1 | 1 | 1 |
| Check if course id matches current | 1 | n | n |
| add node | 1 | n | n |
| **Check if prereq exists** | 1 | n | n |
| **Add prereq** | 1 | n | n |
| if course id is lower than current | 1 | n | n |
| Set current to left node | 1 | n | n |
| set current to right node | 1 | n | n |
| **Total Cost** | | | 7n+2 |
| **Runtime** | | | O(N) |

Based on the runtime analysis vectors would be the best structure for sorting course data. Since the O notation of each structure is O(N) we can compare runtime by looking at the total cost. With fewer lines to compile vector sort is the fastest. However, speed isn’t the only consideration. We should remember that course information can change regularly. If a data structure is hard to edit that will decrease its practical efficiency.

If we compare speed and ease of use vectors would be the data structure I recommend. The fact that hash tables are sorted during creation is handy, but risks collisions during editing. Binary trees can be sorted quickly, but still may not be best for this situation. Losing a small amount of time while sorting is made up for by the ability to print courses quickly.