Thomas Bartlett

CS 300 – H7099

Project 1

8/5/2022

**//Defines structure to hold course info (used for vector, hash table, and BST)**

struct Course{

define string courseNum, courseName

define vector<string> PreReq

}

**//Open, read, and parse csv file into vector**

vector<Course> loadCourses{

define vector<Course> courses

define string row, token

initialize ifstream inFS

open file with inFS(file name/path)

if (file is not open)

output “Could not open file (file name/path)”

return -1

while (getline(inFS, row)){

initialize int count = 0

initialize stringstream line

create Course object course

while (line.good()){

getline(line, token, ‘,’)

if (count is 0)

token is course.courseNum

++count

else if (count is 1)

token is course.courseName

++count

else

if (token is a courseNum in courses vector)

push token to back PreReq vector

++count

else

output “Prerequisite has to be a previous course”

++count

}

if (count is less than 2)

output “Error! Must have a course number and course name.”

clear line

push course object to back of courses

}

close file

return courses

}

**//search for and print course information using Vector**

void printCourseInfo (Vector<Course> courses, string courseNumber){

for (iterate through courses){

if (courseNumber is the same as a courseNum in courses){

print out course information

if (course has prerequisite(s)){

print prerequisite course information

}

}

}

}

**//print all courses in alphanumeric order using vector**

void printAllCourses (Vector<Course> courses){

sort vector courses

for(iterate through courses){

print out course information

if (course has prerequisite(s)){

print prerequisite course information

}

}

}

**//Open, read, and parse csv file into Hash Table**

Hashtable<Course> loadCourses{

define Hashtable <Course> courses

define string row, token

initialize ifstream inFS

open file with inFS(file name/path)

if (file is not open)

output “Could not open file (file name/path)”

return -1

while (getline(inFS, row)){

initialize int count = 0

initialize stringstream line

create Course object course

while (line.good()){

getline(line, token, ‘,’)

if (count is 0)

token is course.courseNum

++count

else if (count is 1)

token is course.courseName

++count

else

if (token is a courseNum in courses vector)

push token to back PreReq vector

++count

else

output “Prerequisite has to be a previous course”

++count

}

if (count is less than 2)

output “Error! Must have a course number and course name.”

clear line

if (HashSearch(hashTable, item⇢key) == null) {

bucketList = hashTable[Hash(item⇢key)]

node = Allocate new linked list node

node⇢next = null

node⇢data = item

ListAppend(courses, node)

}

}

close file

return courses

}

**//search for and print course information using Hash Table**

void printCourseInfo (Hashtable<Course> courses, string courseNumber){

bucketList = hashTable[Hash(courseNumber)]

itemNode = ListSearch(courses, courseNumber)

if (itemNode is not null)

print out course information

if (course has prerequisite(s)){

print prerequisite course information

}

else

return null

}

**//print all courses in alphanumeric order using hash table**

void printAllCourses (Hashtable<Course> courses){

sort hashtable courses

for(iterate through courses){

print out course information

if (course has prerequisite(s)){

print prerequisite course information

}

}

}

**//Open, read, and parse csv file into BST**

Tree<Course> loadCourses{

define Tree <Course> courses

define string row, token

initialize ifstream inFS

open file with inFS(file name/path)

if (file is not open)

output “Could not open file (file name/path)”

return -1

while (getline(inFS, row)){

initialize int count = 0

initialize stringstream line

create Course object course

while (line.good()){

getline(line, token, ‘,’)

if (count is 0)

token is course.courseNum

++count

else if (count is 1)

token is course.courseName

++count

else

if (token is a courseNum in courses vector)

push token to back PreReq vector

++count

else

output “Prerequisite has to be a previous course”

++count

}

if (count is less than 2)

output “Error! Must have a course number and course name.”

clear line

if (courses⇢root == null) {

courses⇢root = new Node(course)

node⇢left = null

node⇢right = null

}

Else{

current = courses⇢root

while (current is not null)

if (node⇢key is less than current⇢key)

if (current⇢left is null)

current⇢left = node

current = null

else

current = current⇢left

else

if (current⇢right is null)

current⇢right = node

current = null

else

current = current⇢right

node⇢left = null

node⇢right = null

}

}

close file

return courses

}

**//search for and print course information using BST**

void printCourseInfo (Tree<Course> courses, string courseNumber){

current = courses⇢root

while(current is not null)

if (courseNumber == current⇢key)

print out course information

if (course has prerequisite(s)){

print prerequisite course information

}

else if (courseNumber < current⇢key)

current = current⇢left

else

current = current⇢right

return null

}

**//print all courses in alphanumeric order using BST**

void printAllCourses (Tree<Course> courses){

sort Tree courses

for(iterate through courses){

print out course information

if (course has prerequisite(s)){

print prerequisite course information

}

}

}

**//main menu**

Initialize int choice to 0

Define string courseID

while(choice is not 9){

//Display menu items 1 per line

Menu

1. Load Courses
2. Display All Courses
3. Find A Course

9. Exit

Enter Choice:

Read in user choice and assign to int choice

Switch(choice){

case 1:

Initialize data structure(vector, hash table, or BST)

call function to load courses into data structure.

Break

case 2:

call function to print all courses in alphanumeric order

break

case 3:

print “Enter course ID:”

read in course ID to string courseID

call function to display course based on user input

break

}

}

Print “goodbye”

**OPENING AND READING LINES OF A FILE**

|  |  |  |  |
| --- | --- | --- | --- |
| **Code** | **Line Cost** | **# Times Executed** | **Total Cost** |
| initialize ifstream inFS | 1 | 1 | 1 |
| open file with inFS | 1 | 1 | 1 |
| if (file is not open) | 1 | N | N |
| output “Could not open file (file name/path)” | 1 | 1 | 1 |
| while (getline(inFS, row)) | 1 | N | N |
| Total Cost | | | 2N + 3 |
| Runtime | | | O(N) |

**CREATING COURSE OBJECTS**

|  |  |  |  |
| --- | --- | --- | --- |
| initialize stringstream line | 1 | 1 | 1 |
| while (line.good()) | 1 | N | N |
| getline(line, token, ‘,’) | 1 | 1 | 1 |
| if (count is 0) | 1 | N | N |
| token is course.courseNum | 1 | 1 | 1 |
| "++count" | 1 | 1 | 1 |
| else if (count is 1) | 1 | N | N |
| token is course.courseName | 1 | 1 | 1 |
| "++count" | 1 | 1 | 1 |
| else | 1 | N | N |
| if (token is a courseNum in courses vector) | 1 | N | N |
| push token to back PreReq vector | 1 | 1 | 1 |
| "++count" | 1 | 1 | 1 |
| else | 1 | N | N |
| output “Prerequisite has to be a previous course” | 1 | 1 | 1 |
| "++count" | 1 | 1 | 1 |
| if (count is less than 2) | 1 | N | N |
| output “Error! Must have a course number and course name.” | 1 | 1 | 1 |
| Total Cost | | | 7N + 11 |
| Runtime | | | O(N) |

**ADVANTAGES AND DISADVANTAGES OF DATA STRUCTURES:**

**Vectors:**

As far as vectors go, if it is a sorted vector, a search function is just as efficient if not more efficient than a hash table or binary search tree (BST). By applying a binary search to the vector, it is actually a better runtime than searching through a hash table and has the same runtime of that of a BST. It may also be preferred to use for smaller lists over BSTs for memory purposes as well. However, once you scale this to let’s say a large data base, or for a project where you may be doing a lot or frequent insertions and deletions, the performance of a vector falls behind that of a BST.

**Hash Tables:**

For hash tables with good or great hashing algorithms, insertion and deletions become an advantage of this data structure. However, if the hashing algorithm is poor, or the size of the table is not accounted for, a lot of collisions can occur making insertions and deletions become more tedious. Hash tables also fall behind when it comes to searching as well when there are a lot of collisions. Another disadvantage is being able to perform sorting on a hash table.

**Binary Search Trees:**

Binary search trees (BSTs) excel when it comes to projects where you are setting something up like a database. While a sorted vector and BST have very similar search runtimes, BSTs are preferred for use when there are large amounts of data, or if you are going to be performing frequent or a lot of insertions/deletions (as mentioned before).

**RECOMMENDATION:**

For this project (even with the small list for purposes of the project for this class) I would recommend using a binary search tree. In a real-world situation for a project like this, it would be expected that there would be hundreds of classes offered for a university. Also, there is the possibility that there may be the need to add or delete classes from the course list in the future. Taking these into account, setting it up with a BST data structure now will save on performance in the future.