CS-300 Project One

Matt Smith

8/3/2023

Vector Pseudocode:

vector<Course> loadCourses(string csvPath) {

print the file name being loaded

create courses vector data structure to hold collection of courses

initialize CSV parser using the file name

if the file opens successfully

for each row in csv file

create course data structure and add to collection of courses

set tokens equal to number of columns in file row

if columns are less than 2 then

print invalid file format

return empty vector

set courseNumber equal to token at index 0

set courseName equal to token at index 1

create empty vector for prerequisites

for each column between 2 and tokens

set prerequisite equal to tokens at current index

push back prerequisite to prerequisites

set course object course number equal to courseNumber

set course object course title equal to courseName

set course object course prerequisites equal to prerequisites

push back course object to courses data structure

return courses

else

print that the file failed to open

return vector

}

struct Course {

define string for course number

define string for course title

define vector string for course prerequisites

}

void displayCourses(vector<Course> courses, string courseNumber){

set courseFound bool equal to false

for each course in courses

if course object id is equal to courseNumber then

set courseFound equal to true

print course number

print course title

if courseFound prerequisite length is greater than zero then

for each prerequisite in courseFound prerequisites

print prerequisite

else

print that no prerequisites were found

break loop

if courseFound is equal to false then

print that the course was not found

}

int main(){

define string for csvPath

define vector for courses

define courseLoaded Boolean and set it equal to false

define variable for user choice and set it to zero

while user choice does not equal 4

print menu options: load courses, print courses, print individual course, and end program

get user input and assign it to user choice

create switch case with user choice as parameter

if user enters 1

call loadCourses function with csvPath and assign it to course

print how many courses were loaded

set coursesLoaded equal to true

if user enters 2

if coursesLoaded is equal to true then

call printCourses function with courses, zero, and courses size minus one

else

print no courses were loaded

if user enters 3

if coursesLoaded is equal to true then

define string for courseId

get user input for coursed

call displayCourses function with courses

else

print no courses were loaded

if user enters 4

print good bye

end program

}

void printCourses(vector<Course>& courses, int begin, int end){

define mid and set to zero

if begin is greater than or equal to end then

for i equals zero and while i is less than size of courses, add one to i

print course id and course title

for each prerequisite in course prerequisites

print prerequisites

}

Hash Table Pseudocode:

void loadCourses(string csvPath, Hashtable\* hashTable) {

print the file name being loaded

initialize CSV parser using the file name

if the file opens successfully

for each row in csv file

create course data structure and add to collection of courses

if length of columns is less than 2 then

print invalid file format

return empty hashTable

set courseNumber equal to token at index 0

set courseName equal to token at index 1

if length of columns is greater than 2 then

for each column between 2 and tokens

set prerequisite equal to tokens at current index

add prerequisite to course

set course object course number equal to courseNumber

set course object course title equal to courseName

set course object course prerequisites equal to prerequisites

insert course object into hashTable

return courses

else

print that the file failed to open

return hashTable

}

struct Course {

define string for course number

define string for course title

define vector string for course prerequisites

}

void displayCourses(Hashtable<Course> courses, String courseNumber) {

if courseNumber is not found in courses then

print that the course was not found

return

set courseList equal to courses at courseNumber

for each course in courseList

print course number and course title

if course has prerequisites, then

print prerequisites

else

print no prerequisites found

}

int main(){

define string for csvPath

define a hash table with variable name courseTable

set courseTable equal to new Hash table

Create course object

define courseLoaded Boolean and set it equal to false

define variable for user choice and set it to zero

while user choice does not equal 4

print menu options: load courses, print courses, print individual course, and end program

get user input and assign it to user choice

create switch case with user choice as parameter

if user enters 1

call loadCourses function with csvPath and courseTable

print how many courses were loaded

set coursesLoaded equal to true

if user enters 2

if coursesLoaded is equal to true then

call printCourses function from courseTable

else

print no courses were loaded

if user enters 3

if coursesLoaded is equal to true then

define string for courseId

get user input for courseId

call displayCourses function from courseTable with parameter courseId

if courseId was found then

print course information

else

print course not found

else

print no courses were loaded

if user enters 4

print good bye

end program

}

void HashTable::printCourses(){

for auto IT is equal to beginning node and does not equal end node, add one to IT

set pointer to IT equal to new node object

if node key does not equal UINT\_MAX

print current node course information

set node equal to next node

while node does not equal nullptr

print current node course information

set node equal to next node

}

Binary Search Tree Pseudocode:

void loadCourses(string csvPath, BinarySearchTree\* bst) {

print the file name being loaded

initialize CSV parser using the file name

create randomCourses vector to hold course information

if the file opens successfully

for each row in csv file

create course data structure and add to collection of courses

if length of columns is less than 2 then

print invalid file format

return empty hashTable

set courseNumber equal to token at index 0

set courseName equal to token at index 1

if length of columns is greater than 2 then

for each column between 2 and tokens

insert prerequisite at current index in course object

insert course object into randomCourses vector

shuffle randomCourses vector

for each item in randomCourses

call insert function with parameter randomCourses at index

return courses

else

print that the file failed to open

return bst

}

struct Course {

define string for course number

define string for course title

define vector string for course prerequisites

}

void displayCourses(Course course) {

print course number and course title

if the size of course prerequisites is greater than zero

for each item in course prerequisites

print prerequisite

return

}

int main(){

define string for csvPath

define a binary search tree to hold courses to variable bst

set bst equal to new binary search tree

define course object

define courseLoaded Boolean and set it equal to false

define variable for user choice and set it to zero

while user choice does not equal 4

print menu options: load courses, print courses, print individual course, and end program

get user input and assign it to user choice

create switch case with user choice as parameter

if user enters 1

call loadCourses function with csvPath and binary search tree

print how many courses were loaded

set coursesLoaded equal to true

if user enters 2

if coursesLoaded is equal to true then

call printCourses function with courses

else

print no courses were loaded

if user enters 3

if coursesLoaded is equal to true then

define string for courseId

get user input for courseId

call displayCourses function with courseId

if courseId was found then

print course information

else

print course not found

else

print no courses were loaded

if user enters 4

print good bye

end program

}

void BinarySearchTree::printCourses(Node\* node) {

if node does not equal nullptr then

call printCourses function with left node

print course information

call printCourses function with right node  
}

**Evaluation**:

| **Vector Loading Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **Print the file name loaded** | 1 | 1 | 1 |
| **Create courses structure to hold collection** | 1 | 1 | 1 |
| **Initialize parser with file name** | 1 | 1 | 1 |
| **If the file opens successfully** | 1 | 1 | 1 |
| **For each row in file** | 1 | n | n |
| **Create course data structure and add to collection of courses** | 1 | n | n |
| **Set tokens equal to number of columns in file** | 1 | n | n |
| **If columns are less than 2 then** | 1 | n | n |
| **Print invalid file format** | 1 | 1 | 1 |
| **Return empty vector** | 1 | 1 | 1 |
| **Set courseNumber equal to token at index 0** | 1 | n | n |
| **Set courseName equal to token at index 1** | 1 | n | n |
| **Create empty vector for prerequisites** | 1 | n | n |
| **For each column between 2 and token** | 1 | n | n |
| **Set prerequisite equal to token at current index** | 1 | n | n |
| **Pushback prerequisite to prerequisites** | 1 | n | n |
| **Set course object course number equal to course number** | 1 | n | n |
| **Set course object course title equal to courseName** | 1 | n | n |
| **Set course object course prerequisites equal to prerequisites** | 1 | n | n |
| **Push back course object to courses data structure** | 1 | n | n |
| **Return courses** | 1 | 1 | 1 |
| **Else** | 1 | 1 | 1 |
| **Print file failed to open** | 1 | 1 | 1 |
| **Return vector** | 1 | 1 | 1 |
| **Total Cost** | | | 14n + 10 |
| **Runtime** | | | O(n) |
| **Vector Structure code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| **Define string for course number** | 1 | 1 | 1 |
| **Define string for course title** | 1 | 1 | 1 |
| **Define vector string for course prerequisites** | 1 | 1 | 1 |
| **Total Cost** | | | 3 |
| **Runtime** | | | O(1) |

| **Hashtable Loading Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **Print the file name loaded** | 1 | 1 | 1 |
| **Initialize parser with file name** | 1 | 1 | 1 |
| **If the file opens successfully** | 1 | 1 | 1 |
| **For each row in file** | 1 | n | n |
| **Create course data structure and add to collection of courses** | 1 | n | n |
| **If columns are less than 2 then** | 1 | n | n |
| **Print invalid file format** | 1 | 1 | 1 |
| **Return empty hashtable** | 1 | 1 | 1 |
| **Set courseNumber equal to token at index 0** | 1 | n | n |
| **Set courseName equal to token at index 1** | 1 | n | n |
| **If length of columns is greater than 2** | 1 | n | n |
| **For each column between 2 and tokens** | 1 | n | n |
| **Set prerequisite equal to token at current index** | 1 | n | n |
| **Add prerequisite to course** | 1 | n | n |
| **Set course object course number equal to course number** | 1 | n | n |
| **Set course object course title equal to courseName** | 1 | n | n |
| **Set course object course prerequisites equal to prerequisites** | 1 | n | n |
| **Insert course object into hashTable** | 1 | n | n |
| **Return courses** | 1 | 1 | 1 |
| **Else** | 1 | 1 | 1 |
| **Print file failed to open** | 1 | 1 | 1 |
| **Return hashTable** | 1 | 1 | 1 |
| **Total Cost** | | | 13n + 9 |
| **Runtime** | | | O(n) |
|  | | |  |
| **Hashtable Structure code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| **Define string for course number** | 1 | 1 | 1 |
| **Define string for course title** | 1 | 1 | 1 |
| **Define vector string for course prerequisites** | 1 | 1 | 1 |
| **Total Cost** | | | 3 |
| **Runtime** | | | O(1) |

| **Tree Loading Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **Print the file name loaded** | 1 | 1 | 1 |
| **Initialize parser with file name** | 1 | 1 | 1 |
| **If the file opens successfully** | 1 | 1 | 1 |
| **For each row in file** | 1 | n | n |
| **Create course data structure and add to collection of courses** | 1 | n | n |
| **If columns are less than 2 then** | 1 | n | n |
| **Print invalid file format** | 1 | 1 | 1 |
| **Return empty tree** | 1 | 1 | 1 |
| **Set courseNumber equal to token at index 0** | 1 | n | n |
| **Set courseName equal to token at index 1** | 1 | n | n |
| **If length of columns is greater than 2** | 1 | n | n |
| **For each column between 2 and tokens** | 1 | n | n |
| **Insert prerequisite at current index in course object** | 1 | n | n |
| **Insert course object into randomCourses vector** | 1 | n | n |
| **Shuffle randomCourses vector** | 1 | 1 | 1 |
| **For each item in randomCourses** | 1 | n | n |
| **Call insert function with parameters randomCourses at index** | 1 | n | n |
| **Return courses** | 1 | 1 | 1 |
| **Else** | 1 | 1 | 1 |
| **Print file failed to open** | 1 | 1 | 1 |
| **Return tree** | 1 | 1 | 1 |
| **Total Cost** | | | 11n + 10 |
| **Runtime** | | | O(n) |

| **Tree Structure code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **Define string for course number** | 1 | 1 | 1 |
| **Define string for course title** | 1 | 1 | 1 |
| **Define string for course prerequisites** | 1 | 1 | 1 |
| **Total Cost** | | | 3 |
| **Runtime** | | | O(1) |

Vector Advantages:

* Easy to implement and use.
* Accessing elements is efficient and takes a constant amount of time.
* Well-suited for storing sorted data with a small number of elements.

Vector Disadvantages:

* Inserting elements can be slow when the vector size exceeds capacity. Requires reallocation and copying of elements.
* Searching for elements must be done through a linear search, which can be slower than other data structures.

Hash Table Advantages:

* Provides fast insertion and retrieval operations.
* Can quickly look up information based on keys, such as course numbers.

Hash Table Disadvantages:

* Can be memory-intensive since they need to allocate space for data and the hash table.
* Collisions can happen, especially with many elements.

Tree Advantages:

* Balanced trees are efficient for maintaining data in sorted order.
* Balanced trees offer reliable performance by ensuring their height remains logarithmic with respect to the number of elements.

Tree Disadvantages:

* Inserting and deleting elements can require rebalancing the tree, which can have a time complexity of O(log n) for the best case, or O(n) in the worst case.
* Implementing a balanced tree can be complex compared to other structures like vectors or hash tables.

Recommendation:

The data structure I plan to use in my code will be the binary search tree. The runtime for each data structure works out to be the same so it comes down to the advantages and disadvantages of each data structure. Considering a hash table can be memory intensive and that possible collisions may happen as more courses are added, I didn’t think that would be the best fit for this project. While vectors are easy to implement and have an access time complexity of O(1), the program will be slower when it comes to insertion and search operations which also doesn’t seem like a good fit. A binary search tree seems like the best choice because of the reliable performance, as long as the tree is balanced. From my pseudocode, I believe that shuffling the courses before loading them into the tree will make it balanced and make it the most efficient choice of the three.