CS 300 Project 1

Parker Harb

Complexity Evaluation:

So first off, we have to evaluate the complexity of reading data into the app. I know from experience, we should be able to keep this around O(N). As for cost per line, we’ll need the following.

String line

Ifstream file (“location.extension”)

readStage = 0

Vector stages = [courseId, name, prereqs]

tempString = “”

While (getline(file, line))

Init new course object

For each character in line:

Check if character is a ‘,’ then

if stages[readStage] is courseID then

courseObject.courseID = tempString

Increment readStage

if stages[readStage] is name then

courseObject.courseName = tempString

Increment readStage

if stages[readStage] is prereq then

Add tempString to courseObject prereq vector

tempString is set to empty

Add character to tempString

Add course to data structure

So actually, since we need to go over each course, which is N length, and there are N courses, we’re actually going to be doing O(N^2). That’s because we’ll be doing N operations within N operations. I could write out all the constants in this analysis, but they aren't really relevant for complexity, as the cost associated with initializing a vector for example is incredibly negligible.

On the subject of the different options we have for data structures, each one has advantages and disadvantages, pros and cons:

So for a vector, it’s generally an easy solution to create, with consistent performance. This is a big plus, as poorly implemented data structures can kill your code. As for performance, it’s only really better if you know the index value you want. For example when you’re iterating down the list, it’s pretty efficient.

For hash tables, they’re very fast. Hard to use though. But very fast. They’re also a really good choice here, as we can look up the location of a course directly with the courseID. Since the index is a string, hash tables appear very attractive.

Binary search trees are also pretty fast, but not as good as hash tables. That being said, they’re auto sorted, which is very nice. The associated costs with sorting with selection sort is O(n^2), this is avoided with binary search.

As for which one I’ll use in my code, I have to go with a binary search tree. Avoiding the costs associated with sorting is really great, and accessing data doesn’t take too long, so the gains won’t be very extreme from using a hash table. That’s my thinking.

**Pseudocode**

For Menu:

Function displayMenu(){  
 Answer = empty string

while answer does not equal 9:

Print instructions + friendly greeting

Input answer

If answer is 1:

loadData

If answer is 2:

printAll

If answer is 3:

displayCourse

If answer is 9

Exit

}

For sorting:

I’ll use a selection sort, as that’s pretty much my go to when I need to sort something.

Function sortCourses(){

For start(i) 0 end course list length -1 and increment by 1

Set min to increment

For start(j) at position plus 1 and end at course list length, increment 1

If course at position j is less then course at min pos:

Set min to j

Swap positions of courses at i and min

}

Function printAll(){  
 sortCourses()

For each course in courseList:

Print courseList at position toString()

}

VECTOR PSEUDOCODE:

Function AccessData(filename){  
 Instantiate file reader object ifstream

Open file location(“location.extension”)

String line

while(getline(filereader, line){

Authenticate line:

Each line starts with the course code, followed by the course name

And if there are prereqs it is then followed by another comma

Per prereq

All of this data must then be plugged into a vector of objects

Vector courseList.push(new course)

}

Ensure that each course prereq exists:

Loop over your vector, use a nested for loop.

For each prereq search the vector for the class

}

Class Course{

String courseName

String courseCode

Vector prereqs

}

Constructor(code,name){

courseCode = code

courseName = name

}

Void addPrereq(prereq){

prereqs.push(prereq)

}

Function SearchForCourse(courseCode){

For each course in courselist:

If course.couseCode == courseCode:

print(toString(course))

}

String function toString(course c){

String text = “Course: ”

Text += c.courseCode + “ - “;

Text += c.courseName + “ - “

For each prereq in vector c.prereqs

Text += “ - Prereq: ”+ prereq + “ - “

Return Text

}

HASH PSEUDOCODE:

Pseudocode milestone

Parker Harb

Function AccessData(filename){  
 Instantiate file reader object ifstream

Open file location(“location.extension”)

String line

while(getline(filereader, line){

Authenticate line: ensure data is correct

Each line starts with the course code, followed by the course name

And if there are prereqs it is then followed by another comma

Per prereq

All of this data must then be plugged into a vector hash of objects

You do this by going character by character within the line, and storing

The data in placeholder variables, which then go into your course object

constructor

insert(course)

}

Ensure that each course prereq exists:

Loop over your vector, use a nested for loop.

For each prereq search the vector for the class

}

Class Course{

String courseName

String courseCode

Vector prereqs

}

Constructor(code,name){

courseCode = code

courseName = name

}

Void addPrereq(prereq){

prereqs.push(prereq)

}

Class hashtable{

Struct node{

Course course

Int Key

Node \*next

node(){

Key = NULL

Next = nullptr

}

node(key, node){

Key = key

Next = get pointer node

}

}

Vector nodes

}

Function printAll(){  
 For each node in hash vector list:

print(toString(node->course)

}

Function remove(courseCode){

Key = hash(courseCode)

Get node at key and delete content

}

Function hash(courseName){

Return toInt(courseCode)/size

}

Function insert(course){

Key = hash(toString(course.courseCode)

if(list at key is not empty){

Replace node course data with new course

}

Else{

Add course to node hash list

}

}

Function SearchForCourse(courseCode){

Key = hash(courseCode)

Return node at key

}

String function toString(course c){

String text = “Course: ”

Text += c.courseCode + “ - “;

Text += c.courseName + “ - “

For each prereq in vector c.prereqs

Text += “ - Prereq: ”+ prereq + “ - “

Return Text

}

I just adapted my previous milestone to use a hashtable. I hope that’s what you wanted, though I suppose I would have organized much of the code the same way as the previous milestone anyway. Hope this is correct.

LINKED LIST PSEUDOCODE:

Pseudocode milestone 5

Parker Harb

Function AccessData(filename){  
 Instantiate file reader object ifstream

Open file location(“location.extension”)

String line

while(getline(filereader, line){

Authenticate line: ensure data is correct

Each line starts with the course code, followed by the course name

And if there are prereqs it is then followed by another comma

Per prereq

All of this data must then be plugged into a binary tree

You do this by going character by character within the line, and storing

The data in placeholder variables, which then go into your course object

constructor

insert(course)

}

Ensure that each course prereq exists:

Loop over your vector, use a nested for loop.

For each prereq search the vector for the class

}

Class Course{

String courseName

String courseCode

Vector prereqs

}

Constructor(code,name){

courseCode = code

courseName = name

}

Void addPrereq(prereq){

prereqs.push(prereq)

}

Struct node{

Course course

Node \*next

node(){

Next = nullptr

}

Class BinarySearchTree{

Node root

Void addNode

Void removeNode

Void InOrder

Course searchForCourse

}

Function inOrder(){  
 Inorder print all courses

}

Function removeNode(courseCode){

Find the node and remove it, then link it’s parent and any possible children to it’s replacement in the list

}

Function addNode(course){

Traverse the tree until you find an empty node and fill it with the course

}

Function SearchForCourse(courseCode){

Traverse tree until you find the node that matches.

}

String function toString(course c){

String text = “Course: ”

Text += c.courseCode + “ - “;

Text += c.courseName + “ - “

For each prereq in vector c.prereqs

Text += “ - Prereq: ”+ prereq + “ - “

Return Text

}