Brandon Kelfstrom

4/8/22

CS 300 Project 1

**Pseudocode for data structures:**

**Vector**

**To open and read a file:**

Initialize fstream variable fin

Initialize hashtable courses

Initialize string line, word

Open CSV file

While there are lines left in file

Getline from file fin

Read line until comma

If there is no comma in line

Print error message

Else

Append string to table line

If more than 1 comma

Check string against each key in hashtable courses

If strings match

Continue

Else

Print error message

**To create course objects:**

Initialize ofstream variable fout

Initialize String course, title, prereq

Initialize vector<string> prereqs

While user does not exit

Get course from user

Get title from user

fout << course << “, “ << title << “, “

Print (“Are there any prerequisites?)

While yes

Get prereq from user

Append prereq to vector prere

Print (“Are there any more?)

Get user input

For each item in prereqs

If last item

fout << item << “/n”

Else

fout << item << “, “

Print (“would you like to exit?”)

**Find a course:**

Get courseRequest from user

Load courses vector

For all courses

If course matches courseRequest

Print course information

For each prereq

Print prereq

Else

Print error message

**Hash table**

**To open and read a file:**

Initialize fstream variable fin

Initialize hashtable courses

Initialize string line, word

Open CSV file

While there are lines left in file

Getline from file fin

Read line until comma

If there is no comma in line

Print error message

Else

Append string to table line

If more than 1 comma

Check string against each key in hashtable courses

If strings match

Continue

Else

Print error message

**Create course Objects:**

Initialize int number of buckets

Initialize String course, title, prereq

Initialize Hashtable

Initialize vector<string> prereqs

While user does not exit

Get course from user

Get title from user

If Hashtable[course] does not exist {

Hashtable[course] course = course

Hashtable[course] title = title

}

Print (“Are there any prerequisites?)

While yes

Get prereq from user

Append prereq to vector prere

Print (“Are there any more?)

Get user input

For each item in prereqs

Prereq = item

Print (“would you like to exit?”)

**Print Information:**

Get courseRequest from user

If Hashtable[courseRequest]does not exist {

Print error

}

Else {

Print hashTable[courseRequest] course

Print hashTable[courseRequest] title

Print hashTable[courseRequest] prereq

**Tree Pseudocode**

**To open and read a file:**

Initialize fstream variable fin

Initialize tree courses

Initialize string line, word

Open CSV file

While there are lines left in file

Getline from file fin

Read line until comma

If there is no comma in line

Print error message

Else

Append string to table line

If more than 1 comma

Check string against each value in tree courses

If strings match

Continue

Else

Print error message

**Create course Objects:**

Initialize String course, title, prereq

Initialize Tree

Initialize vector<string> prereqs

While user does not exit

Get course from user

Get title from user

If Tree[course] does not exist {

Push course to Tree

Push title to Tree

}

Print (“Are there any prerequisites?)

While yes

Get prereq from user

Append prereq to vector prere

Print (“Are there any more?)

Get user input

For each item in prereqs

Push item to Tree

Print (“would you like to exit?”)

**Print Information:**

Get courseRequest from user

If Treee[courseRequest]does not exist {

Print error

}

Else {

For i = 0; i < depth, i++ {

Print Tree item

}

}

**Pseudocode for menu:**

Int i = 0

While i is not 1 {

Print menu header

Print menu elements

Get user input

Switch input {

Case 1:

Call Load Data Structure function

Break

Case 2:

Call Alphanumeric Print function

Break

Case 3:

Get user input course

Call print course function

Break

Case 4:

Set i equal to 1

Break

}

}

**Pseudocode for alphanumeric sorting:**

(This will sort 0-9, a-z)

Initialize compareString as empty string

Initialize list sorted

For each course in courses {

If character 0 of course is greater than character 0 of compareString {

Insert course after compareString in list sorted

}

Else If character 0 of course is equal to character 0 of compareString {

For each character in course {

If character is not equal to character x of compareString {

Insert before if less than

Insert after if greater than

}

}

Else {

Insert Course before compareString in list sorted

}

}

For Each item in list sorted {

Print item

}

**Evaluation**

Run-time analysis:

Vector:

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| While user does not exit | 1 | n | n |
| Get course from user | 1 | 1 | 1 |
| Get title from user | 1 | 1 | 1 |
| fout << course << “, “ << title << “, “ | 1 | 1 | 1 |
| Print (“Are there any prerequisites?) | 1 | 1 | 1 |
| While yes | 1 | n | n |
| Get prereq from user | 1 | 1 | 1 |
| Append prereq to vector prere | 1 | 1 | 1 |
| Print (“Are there any more?) | 1 | 1 | 1 |
| Get user input | 1 | 1 | 1 |
| For each item in prereqs | 1 | n | n |
| If last item | 1 | 1 | 1 |
| Else | 1 | n | n |
| fout << item << “, “ | 1 | 1 | 1 |
| Print (“would you like to exit?”) | 1 | 1 | 1 |
| **Total Cost** | | | 4n + 10 |
| **Runtime** | | | O(n) |

Hashtable:

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| While user does not exit | 1 | n | n |
| Get course from user | 1 | 1 | 1 |
| Get title from user | 1 | 1 | 1 |
| If Hashtable[course] does not exist { | 1 | n | n |
| Hashtable[course] course = course | 1 | 1 | 1 |
| Hashtable[course] title = title | 1 | n | n |
| Print (“Are there any prerequisites?) | 1 | 1 | 1 |
| While yes | 1 | n | n |
| Get prereq from user | 1 | 1 | 1 |
| Append prereq to vector prere | 1 | 1 | 1 |
| Print (“Are there any more?) | 1 | 1 | 1 |
| Get user input | 1 | 1 | 1 |
| For each item in prereqs | 1 | n | n |
| Prereq = item | 1 | 1 | 1 |
| Print (“would you like to exit?”) | 1 | 1 | 1 |
| **Total Cost** | | | 5n + 10 |
| **Runtime** | | | O(n) |

Tree:

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| While user does not exit | 1 | n | n |
| Get course from user | 1 | 1 | 1 |
| Get title from user | 1 | 1 | 1 |
| If Tree[course] does not exist { | 1 | n | n |
| Push course to Tree | 1 | 1 | 1 |
| Push title to Tree | 1 | n | n |
| Print (“Are there any prerequisites?) | 1 | 1 | 1 |
| While yes | 1 | n | n |
| Get prereq from user | 1 | 1 | 1 |
| Append prereq to vector prere | 1 | 1 | 1 |
| Print (“Are there any more?) | 1 | 1 | 1 |
| Get user input | 1 | 1 | 1 |
| For each item in prereqs | 1 | n | n |
| Push item to tree | 1 | 1 | 1 |
| Print (“would you like to exit?”) | 1 | 1 | 1 |
| **Total Cost** | | | 5n + 10 |
| **Runtime** | | | O(n) |

Based on my pseudocode, the hashtable and tree have a similar cost to run while the vector has a lower cost. The vector has 1 less variable cost which could lead to a fair reduction in overall time and hardware energy. The hashtable is easier to store values paired with information. The key system makes searching easier in other functions and also simplifies getting the title and prerequisites since they are all held in one data structure. While the tree structure is easily navigable, the hashtable’s ability to store multiple pieces of information with a key makes it the best data structure for this project.