SNHU CS-300

Sequoia Sutherland

April 8,2023

Project One

// **Vector**

// Create vector to store course info

New empty vector Courses

// Open file to parse

Open file (file name passed by user)

// Read file and store data in Courses vector after checking for CSV format

If file can’t be split on commas:

Return error for incorrect file type

While not at end of file:

For each line in file:

Split data in file at commas

If less than 3 data pieces per line return error

Else:

Create new Courses item and save course number (key), course title, prerequisite 1, prerequisite 2, … prerequisite n

// Search Courses vector and print course info

Prompt user to enter course number to search for

Search Courses vector (course number entered by user)

If course number not in Courses vector:

Return error for course not found

Prompt user to enter another course

Else:

Print course title and prerequisites for course number

// **Hash table**

// Create hash table to store course prerequisite info

New empty hash table Course

// Open file to parse

Open file (file name passed by user)

// Read file and store data in Course hash table after checking for CSV format

If file can’t be split on commas:

Return error for incorrect file type

While not at end of file:

For each line in file:

Split data in file at commas

If less than 2 data pieces per line return error

Else:

Create new Course item and hash and save course number (key), course title, prerequisite 1, prerequisite 2, … prerequisite n

// Validate prerequisites as existing courses

For integer i starting at 0 until the end of the hash table array:

For prerequisite courses at course in place i:

If prerequisite can be found in hash table as a key:

Continue to next prerequisite

Else:

Return error for ineligible prerequisite

// Print course info and prerequisites

For integer j starting at 0 until the end of the hash table array:

Find hash key at j and print key (course number) course title and prerequisites for course

// **Binary search tree**

// Create binary search tree to store course prerequisite info

New tree Course with null root

// Open file to parse

Open file (file name passed by user)

// Read file and store data in Course BST after checking for CSV format

If file can’t be split on commas:

Return error for incorrect file type

While not at end of file:

For each line in file:

Split data in file at commas

If less than 2 data pieces per line return error

Else:

If root is null:

Create new Course root node and save course number, course title, prerequisite 1, prerequisite 2, … prerequisite n as node data

If last three characters of course number are less than root node’s last three of course number:

Recurse down left sub-tree to place Course node

Create new Course root node and save course number, course title, prerequisite 1, prerequisite 2, … prerequisite n as node data

If last three characters of course number are greater than root node’s last three of course number:

Recurse down right sub-tree to place Course node

Create new Course root node and save course number, course title, prerequisite 1, prerequisite 2, … prerequisite n as node data

// Validate prerequisites as existing courses

Starting at root node and traversing BST in order:

For each node traversed:

If prerequisite can be found by searching BST in order for matching Course node:

Continue to next prerequisite

Else:

Return error for ineligible prerequisite

// Print course info and prerequisites

Starting at root node and traversing BST in order:

Print node’s data (course number, course title and prerequisites for course)

// **Menu**

// Print menu options

Print the following strings with newline after each

“1. Print course list”

“2. Print course info”

“3. Exit program”

// Accept user choice and call menu choices

Print the following string and call corresponding menu choice

“Enter menu choice”

While user choice isn’t 3:

If user choice is 1:

Call function to create vector and load with courses

Call function to print all courses from courses vector

If user choice is 2:

Call function to create hash table and load with courses

Print following string and accept user input

“What course would you like to search?”

Call function to search hash table and search for course matching user input

If course found:

Print course info

Else:

Print the following string and then exit

“Course not found”

If user choice is 3:

Exit program

Else:

Print the following string and then accept user input and return to beginning of loop

“Invalid choice, please enter new choice”

// **Print courses in alphanumeric order**

// Create binary search tree to store course prerequisite info

New tree Course with null root

// Open file to parse

Open file (file name passed by user)

// Read file and store data in Course BST after checking for CSV format

If file can’t be split on commas:

Return error for incorrect file type

While not at end of file:

For each line in file:

Split data in file at commas

If less than 2 data pieces per line return error

Else:

If root is null:

Create new Course root node and save course number, course title, prerequisite 1, prerequisite 2, … prerequisite n as node data

If course number is less than root node’s course number:

Recurse down left sub-tree to place Course node

Create new Course root node and save course number, course title, prerequisite 1, prerequisite 2, … prerequisite n as node data

If course number is greater than root node’s course number:

Recurse down right sub-tree to place Course node

Create new Course root node and save course number, course title, prerequisite 1, prerequisite 2, … prerequisite n as node data

// Validate prerequisites as existing courses

Starting at root node and traversing BST in order:

For each node traversed:

If prerequisite can be found by searching BST in order for matching Course node:

Continue to next prerequisite

Else:

Return error for ineligible prerequisite

// Print course info and prerequisites

Starting at root node and traversing BST in ascending order:

Print node’s data (course number, course title and prerequisites for course), new line

// **Vector Runtime Analysis**

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **for all courses** | 1 | n | n |
| **if the course is the same as courseNumber** | 1 | n | n |
| **print out the course information** | 1 | 1 | 1 |
| **for each prerequisite of the course** | 1 | n | n |
| **print the prerequisite course information** | 1 | n | n |
| **Total Cost** | | | 4n + 1 |
| **Runtime** | | | O(n) |

// **Hash Table Runtime Analysis**

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **for all courses** | 1 | n | n |
| **if the course is the same as courseNumber** | 1 | n | n |
| **print out the course information** | 1 | n | n |
| **for each prerequisite of the course** | 1 | n | n |
| **print the prerequisite course information** | 1 | n | n |
| **Total Cost** | | | 5n |
| **Runtime** | | | O(n) |

// **Binary Search Tree Runtime Analysis**

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **for all courses** | 1 | n | n |
| **if the course is the same as courseNumber** | 1 | Log n | Log n |
| **print out the course information** | 1 | n | n |
| **for each prerequisite of the course** | 1 | n | n |
| **print the prerequisite course information** | 1 | n | n |
| **Total Cost** | | | 4n + log n |
| **Runtime** | | | O(n) |

// **Recommendation**

Each structure has benefits depending on what the purpose of the function calling the structure will be. Vectors are simple and would be very useful in the use case of simply storing all of the course information in a structure which can later be accessed. The vector won’t be ordered, so this will cause longer runtimes if searching the vector. A hash table is more complicated to setup than a vector, but if it will be used to search for a particular course and print the course’s information, this is a good choice. Assuming a hash algorithm is carefully made to avoid duplicate keys, this structure can provide a quick and efficient runtime. Binary search trees in this application provide the best option for creating a structure which is sorted from its creation, allowing it to be searched quickly or traversed in a particular order quickly. It can also be added to without resizing.

I would recommend the use of a binary search tree structure for this project because it is likely that courses will often need to be accessed in order, whether to add or remove, or simply print all courses. This structure will provide the university with a method of optimizing runtime while capturing all needed functionality.