Michelle Darling

CS-300 Data Structures and Algorithms: Analysis and Design

11.28.2023

Southern New Hampshire University

**6-2 Project One**

**Vector Data Structure Pseudocode**

1. Design pseudocode to define how the program opens the file, reads the data from the file, parses each line, and checks for file format errors.

CREATE vector for opening files

INITIALIZE loadFile function

INITIALIZE line from file

OPEN fstream file

IF file does not open

PRINT file error message

RETURN

ELSE

WHILE next line exists

INPUT each line

IF line is greater than or equal to TWO (2) parameters

CONTINUE

PUSH line back

ELSE

PRINT error at line

CONTINUE

CLOSE file

RETURN

2. Design pseudocode to show how to create course objects and store them in the appropriate data structure.

CREATE vector for course objects

INITIALIZE vector

INITIALIZE string for each word from line

WHILE there is a new line

ADD new object to vector

PARSE courNum(int), courTitle(string), courPrereq(int) from line

ADD courNum, courTitle, courPrereq to new object

WHILE there is a courPrereq course object

IF courPrereq is equal to courTitle

CONTINUE

ELSE

PRINT error message at courPrereq object

CLOSE file

3. Design pseudocode that will search the data structure for a specific course and print out course information and prerequisites.

INITIALIZE printCour function

FOR all courses

IF course is equal to courNum

PRINT course object

FOR each courPrereq of course object

PRINT each courPrereq course object

ELSE

PRINT course not found error message

RETURN

**Hash Table Data Structure Pseudocode**

1. Design pseudocode to define how the program opens the file, reads the data from the file, parses each line, and checks for file format errors.

INITIALIZE loadFile function

OPEN fstream file

IF file does not open

PRINT file error message

RETURN

ELSE

WHILE nextLine exists

INPUT each line

IF line is greater than or equal to TWO (2) parameters

CONTINUE

PUSH line back

ELSE

OUTPUT error at line

CLOSE file

RETURN

2. Design pseudocode to show how to create course objects and store them in the appropriate data structure.

CREATE vector for course nodes

CREATE HashTable node structure to hold courses

CREATE key for given course

RETRIEVE node using key

IF node equals nullptr

INSERT new course node

PARSE courNum(int), courTitle(string), courPrereq(int) from line

ADD courNum, courTitle, courPrereq to course node

ELSE

WHILE next node is not equal to nullptr

node is equal to next node

next node is equal to new course node

WHILE there is a course node

IF node contains courPrereq

WHILE there is a course node

IF courPrereq is equal to courTitle

CONTINUE

ELSE

PRINT error message at courPrereq object

ELSE

CONTINUE

CLOSE file

3. Design pseudocode that will print out course information and prerequisites.

INITIALIZE printCour function

FOR all nodes

IF searched node is equal to course node

PRINT key, courNum, courTitle, courPrereq

ELSE

PRINT error message

CLOSE file

**Tree Data Structure Pseudocode**

1. Design pseudocode to define how the program opens the file, reads the data from the file, parses each line, and checks for file format errors.

INITIALIZE loadFile function

OPEN fstream file

IF file does not open

PRINT error message

RETURN

ELSE

WHILE nextLine exists

IF line is greater than or equal to TWO (2) parameters

CONTINUE

PUSH line back

ELSE

OUTPUT error at line

CLOSE file

RETURN

2. Design pseudocode to show how to create course objects and store them in the appropriate data structure.

WHILE the file is NOT EOF

FOR each line

FOR first (courNum) and second (courTitle) value

ADD course node

IF node DOES NOT equal nullptr AND current node is greater than added

course node

IF left node is equal to nullptr

node to the left is added course node

RETURN

ELSE

CONTINUE down left node

ELSE

IF right node is equal to nullptr

node to the right is added course node

RETURN

ELSE

CONTINUE down right node

WHILE there is a course node

IF node contains courPrereq

WHILE there is a course node

IF courPrereq is equal to courTitle

CONTINUE

ELSE

PRINT error message at courPrereq object

CLOSE file

3. Design pseudocode that will print out course information and prerequisites.

current node is equal to root

INITIALIZE printCour function

WHILE current node DOES NOT equal nullptr

IF current node is equal to searched courNum

RETURN current courNum

ELSE IF current node is less than searched courNum

current node is equal to current node to the left

ELSE

current node is equal to current node to the right

**Pseudocode Menu**

Create pseudocode for a menu.

WHILE menuOption does not equal 9

PRINT menu

GET user input into menuOption

SWITCH menuOption

CASE 1:

GET user input into fileName

CALL loadFile function from fileName

CLEAR menuOption

BREAK

CASE 2:

CALL courSort function with course objects, 0, and courses size minus 1 variables

CALL printAll function

BREAK

CASE 3:

CALL printCour function

BREAK

CASE 9:

PRINT goodbye message

EXIT program

DEFAULT:

PRINT invalid error message

BREAK

**Alphanumeric Pseudocode**

Design pseudocode that will print out the list of the courses in the Computer Science program in alphanumeric order.

INITIALIZE partition function with course objects, begin, and end variables

INITIALIZE integer low as begin

INITIALIZE integer high as end

INITIALIZE pivot as end minus begin added to begin and divided by two

INITIALIZE boolean done as false

WHILE not done

WHILE course object low as compared to pivot is less than zero

ADD to low variable

WHILE course object pivot as compared to high is less than zero

SUBTRACT from high variable

IF low is greater than or equal to high

done is equal to true

ELSE

SWAP course object low and course object high

ADD to low variable

SUBTRACT from high variable

RETURN high

INITIALIZE courSort function with course objects, begin, and pivot variables

IF course objects size is less than one

PRINT no courses found error message

ELSE IF course objects is equal to one

PRINT course object

ELSE

IF begin is less than pivot

pivot is equal to calling the partition function with course objects, begin, and pivot variables

CALL the courSort function with course objects, begin, and newPivot minus one variables

CALL the courSort function with course objects, newPivot minus one, and pivot variables

**Evaluation**

Evaluate the run-time and memory of data structures that could be used to address the requirements.

| **Vector Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| Create vector for loadFile | 1 | 1 | 1 |
| Initialize line from file | 1 | 1 | 1 |
| Open fstream file | 1 | 1 | 1 |
| If file does not open | 1 | 1 | 1 |
| Print error message | 1 | 1 | 1 |
| Return | 1 | 1 | 1 |
| Else | 1 | 1 | 1 |
| While next line exists | 1 | N | N |
| Input each line | 1 | N | N |
| If line is greater than or equal to two parameters | 1 | N | N |
| Continue | 1 | N | N |
| Else | 1 | N | N |
| Print error at line | 1 | N | N |
| Continue | 1 | N | N |
| Close file | 1 | 1 | 1 |
| Return | 1 | 1 | 1 |
|  |  |  |  |
| Create vector for course objects | 1 | 1 | 1 |
| Initialize vector | 1 | 1 | 1 |
| Call loadFile function | 1 | 1 | 1 |
| Initialize line from file | 1 | 1 | 1 |
| Initialize courNum | 1 | 1 | 1 |
| Initialize courTitle | 1 | 1 | 1 |
| Initialize courPrereq | 1 | 1 | 1 |
| While there is a new line | 1 | N | N |
| Add courNum from line | 1 | N | N |
| Add courTitle from line | 1 | N | N |
| Add courPrereq from line | 1 | N | N |
| While there is a courPrereq from vector object | 1 | N | N |
| If courPrereq is equal to courNum | 1 | N | N |
| Continue | 1 | N | N |
| Else | 1 | N | N |
| Print error message at courPrereq | 1 | N | N |
| Close file | 1 | 1 | 1 |
| Return | 1 | 1 | 1 |
| **Total Cost** | | | 16N + 18 |
| **Runtime** | | | O(N) |

| **Hash Table Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| Initialize loadFile function | 1 | 1 | 1 |
| Open fstream file | 1 | 1 | 1 |
| If file does not open | 1 | 1 | 1 |
| Print error message | 1 | 1 | 1 |
| Return | 1 | 1 | 1 |
| Else | 1 | 1 | 1 |
| While next line exists | 1 | N | N |
| Input each line | 1 | N | N |
| If line is greater than or equal to two parameters | 1 | N | N |
| Continue | 1 | N | N |
| Else | 1 | N | N |
| Print error at line | 1 | N | N |
| Continue | 1 | N | N |
| Close file | 1 | 1 | 1 |
| Return | 1 | 1 | 1 |
|  |  |  |  |
| Create vector for course nodes | 1 | 1 | 1 |
| Create HashTable structure to hold courses | 1 | 1 | 1 |
| Call loadFile function | 1 | 1 | 1 |
| Create key for given course | 1 | 1 | 1 |
| Retrieve node using key | 1 | 1 | 1 |
| If node equals nullptr | 1 | 1 | 1 |
| Insert new node | 1 | 1 | 1 |
| Initialize courNum to new node | 1 | 1 | 1 |
| Initialize courTitle to new node | 1 | 1 | 1 |
| Initialize courPrereq to new node | 1 | 1 | 1 |
| Else | 1 | 1 | 1 |
| While next node is not equal to nullptr | 1 | N | N |
| Node is equal to next node | 1 | N | N |
| Next node is equal to new course node | 1 | N | N |
| While there is a course node | 1 | N | N |
| If node contains courPrereq | 1 | N | N |
| While there is a course node | 1 | N^2 | N^2 |
| If courPrereq is equal to courNum | 1 | N^2 | N^2 |
| Continue | 1 | N^2 | N^2 |
| Else | 1 | N^2 | N^2 |
| Print error message at courPrereq object | 1 | N^2 | N^2 |
| Else | 1 | N | N |
| Continue | 1 | N | N |
| Close file | 1 | 1 | 1 |
| **Total Cost** | | | 5N^2 + 14N + 20 |
| **Runtime** | | | O(N^2 + N) |

| **Binary Search Tree Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| Initialize line from file | 1 | 1 | 1 |
| Open fstream file | 1 | 1 | 1 |
| If file does not open | 1 | 1 | 1 |
| Print error message | 1 | 1 | 1 |
| Return | 1 | 1 | 1 |
| Else | 1 | 1 | 1 |
| While next line exists | 1 | N^2 | N^2 |
| Input each line | 1 | N^2 | N^2 |
| If line is greater than or equal to two parameters | 1 | N^2 | N^2 |
| Continue | 1 | N^2 | N^2 |
| Else | 1 | N | N |
| Print error at line | 1 | N | N |
| Continue | 1 | N | N |
| Close file | 1 | 1 | 1 |
| Return | 1 | 1 | 1 |
|  |  |  |  |
| WHILE the file is not EOF | 1 | N | N |
| For each line | 1 | N | N |
| For first and second value | 1 | N | N |
| Add course node | 1 | N | N |
| Add courNum to node | 1 | N | N |
| Add courTitle to node | 1 | N | N |
| If course node is greater than or equal to three values | 1 | N | N |
| Add courPrereq | 1 | N | N |
| If node does not equal nullptr and current node is greater than added course node | 1 | N | N |
| If left is equal to nullptr | 1 | N | N |
| Left node is course node | 1 | N | N |
| Else | 1 | N | N |
| Continue down left node | 1 | N | N |
| Else | 1 | N | N |
| If right node is equal to nullptr | 1 | N | N |
| Right node is course node | 1 | N | N |
| Else | 1 | N | N |
| Continue down right node | 1 | N | N |
| While there is a course node | 1 | N^2 | N^2 |
| If node contains courPrereq | 1 | N^2 | N^2 |
| While there is a course node | 1 | N^3 | N^3 |
| If courPrereq is equal to courNum | 1 | N^3 | N^3 |
| Continue | 1 | N^3 | N^3 |
| Else | 1 | N^3 | N^3 |
| Print error message at courPrereq object | 1 | N^3 | N^3 |
| Close file | 1 | 1 | 1 |
| **Total Cost** | | | 5N^3 + 6N^2 + 21N + 9 |
| **Runtime** | | | O(N^3 + N^2 + N) |

Based on the advisor’s requirements, analyze each data structure (vector, hash table, and tree). Explain the advantages and disadvantages of each structure in your evaluation.

The vector data structure is probably the simplest to implement while also allowing the ability to quickly load and store the courses into the system. The downfall to this is when it is time to sort the various courses and when searching for a specific course.

The hash table data structure also loads files quickly, and as an advantage also allows for quicker retrieval of searched objects due to the hash table key. The trouble with the hash table data structure, however, is that it is more difficult to implement, and takes a bit longer when it comes to sorting and printing.

For the binary search tree, overall it has the longest runtime as well as being the most complex to set up. On the flip side, once the file has been loaded, it takes no time at all to sort the courses, as the courses are sorted upon being added to the tree. Due to the fact that the tree is automatically sorted upon loading into the data structure, the search and print functions are limited to how long it takes to run through the height of the tree.

Now that you have analyzed all three data structures, make a recommendation for which data structure you will plan to use in your code.

I plan to use the binary search tree to implement my code. While the run time is the longest and the code itself might be a little more complex than the vector and hash table data structures, once the program is up and running it will be the quickest way for the users from ABCU to be able to search and view courses, regardless of whether the user is looking for a specific course, or to be able to look at the entire course list by course number.