Mary Watts

CS 300

Professor Andujar

February 12, 2023

Pseudocode for project 1

Pseudocode 1, resubmittal of all previous code ( vector, hash table, and tree)

Pseudocode for vector:

Struct Course

Initialize courseNumber, course Name

Initialize a vector for the prerequisites

Int numPrerequisiteCourse(vector<Course> course, Course()

Initialize the total of prerequisites

For each prerequisites in totalPrerequisites

Add prerequisite to the totalPrerequisites

1)Pseudocode to read and confirm file

Open file using fstream

Call to open file

If value of file is -1

Output file not found

Else

While end of file has not been reached

Read each line

If less than 2 values in a line

Return output error

Else

Read parameter

If there is a at least 3 parameters

If one of the parameters is equal to the first parameter

Continue

Else

Return output error

Close file

2) Pseudocode for creating courses

Initialize a course vector<string> course info

Open file

Read file

While end of file has not been reached

For each line of the file

For the first and second values

Pushback value into vector

If a third value exists

Push back value until it is on a new line

Close file

3) Pseudocode for searching the data structure

Get input from user

Search for input in vector

If input is in the vector

Output course info

For each pre-requisite of course

Print the pre-requisite course information

Pseudocode for Hash table

Open, read, parse and check file:

Open file with fstream

Call for open of file

If return value is -1

Return file not found

Else

While end of file is not reached

If there are less than 2 values per line

Return error

Else

Read the parameters

If there are 3 or more parameters

If the third parameter is in the first parameter

Continue

Else

Return error

Close file

Create a course objects HashTable:

Initialize a course vector<Nodes> nodes

Create a HashTable class

Create Insert method

Insert items into HashTable

While the end of file has not been reached

For each line of the file

For the first and second parameter

Create a temp holder

If third parameter exists

Add to current value

Call the insert method for each value

Print from HashTable:

Get Input

Set the key to the input received

If key is found

Output the course information

For each prerequisite for the course

Output the prerequisite course information

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

Pseudocode for Binary Trees

1. How to open read and parse each line

Open file

Call to open the file

If returned value is -1

Return error

Else

If it is not the end of the file

If there are fewer than two values on the line

Return error

Else

Read the line

If there are 3 or more values in the line

If the third value is found in the first two values

Continue

Else

Return error

Close file

1. Initialize the course structures and store in tree

Initialize a course struct

Loop through file

If end of file is not reached

For each line in the file

For the first two values

Add course Id and course name

For any value past the first two

Add to prerequisite vector until null is reached

Create tree

Define the binary tree

Create a null root

Create the insert method

If root is null

Current course is root

Else if the course id is larger root

Add to right node

If right is empty

Add course id

Else

If course id is less than the root

Add to the left side

Else

Add to the right side

Else if the course id is smaller than the root

Add to the left side

If the left is empty

The left node becomes the course id

Else

If the course id is less than the root

Add to the left side

Else

Add to the right side

1. Print method

Ask for course information

create the print method

if the root is not empty

search left

if found

print information

search right

if found

print information

------------------------------------------------------------------------------------------------

2 Pseudocode for the Menu

Get input from user, assign it to a variable “input”

create a holder for the Bid value, assign it to the variable “Bid”

while input is not 4

output “1. Load Data Structure”

output “2. Course List”

output “3. Course”

output “4. Exit”

switch(input)

case 1:

load bids

break;

case2:

Print Course List

Break;

Case 3:

Print Course

Break;

Case 4:

Print “Thank you”

Exit program

Break

Default:

Output “Error, no input”

Break;

Pseudocode 3: Vector

Create printSorted method

Create a method to partition the list to do a quick sort with the variables(vector<Courses>, int begin, int end)

Create integer variables pivot, low, high, mid, and temp

Set the low to begin and the high to end

Set mid to (low +((high-low)/2))

Set pivot to mid

If pivot < high

--high

Swap the lower and higher values around the pivot

Set temp to low

Set low to high

Set high to temp

Create a method called quicksort with the variables (vector<Course> int begin, int end)

Set mid to zero

Set low to begin

Set high to end

If low > high

Return

Else

Set low to partition(courses, low, high

Then quicksort(courses, low, high)

quicksort(courses, low+1, high)

Create displayCourse method using the course asked for

Output the courseId, course name, and the course prerequisites if any

Loop through the vector to display the courses

For(the length of the course vector)

Use the displayCourse method to display all courses

Then Create an in order method for the binary tree

If the node is not null

Check the left most side all the way down

inOrder(node->left)

output course information

inOrder(node->right)

output course information

Run time analysis for reading files and creating course objects

|  |  |  |  |
| --- | --- | --- | --- |
| Vector | Line Cost | Executions | Total time |
| Create the vector | 1 | 1 | 1 |
| Each line in the file | 1 | n | n |
| Create item in vector | 1 | N | N |
| With prereq | 1 | n | n |
| Append prereq | 1 | n | n |
| Pushback | 1 | N | N |
|  |  | Total | 5n+1 |
|  |  | Run Time | O(n) |

|  |  |  |  |
| --- | --- | --- | --- |
| Hash Table | Line Cost | Execution | Total time |
| Table Creation | 1 | 1 | 1 |
| Insert method | 0 | 0 | 0 |
| Create Key | 1 | n | n |
| If no Entry for key | 1 | n | n |
| Assign node for key | 1 | n | n |
| else | 1 | n | n |
| Switch around node pointer | 4 | n | 4n |
| else | 1 | n | n |
| Find next opening | 1 | n | n |
| Add new node | 1 | n | n |
| For each new line | 1 | n | n |
| Create vector | 1 | n | n |
| If pre req exists | 1 | n | n |
| Append prereq | 1 | n | n |
| Insert item | 1 | n | n |
|  |  | Total | 16n+1 |
|  |  | Run Time | O(n) |

|  |  |  |  |
| --- | --- | --- | --- |
| Binary Tree | Line Cost | Execution | Total time |
| Add node method | 0 | 0 | 0 |
| If root is null add root | 1 | 1 | 1 |
| If node is less than root add to left | 1 | n | n |
| If no left node | 1 | n | n |
| Become left node | 1 | n | n |
| If node is more than root add right | 1 | n | n |
| If no right | 1 | n | n |
| Become right node | 1 | n | n |
| For each line | 1 | n | n |
| Create vector item | 1 | n | n |
| If prereq exists | 1 | n | n |
| Append prereq | 1 | n | n |
| Insert item | 1 | n | n |
|  |  | Total | 11n+1 |
|  |  | Run Time | O(n) |

---------------------------------------------------------------------------------------------------------------------

Evaluation:

There are pros and cons of each data structure. While vectors are the slowest at searching, having the quick sort option is extremely useful and makes up for the loss of time for the search. Vectors are also the fastest at reading files and putting them into the data structure. Hash tables are great at searching fast using the key to make everything even more speedy, but because of this it also makes making the first list even more time consuming. Hash tables do not allow for a second sorting though so any specific sorting would be painfully slower than the other two options. Binary trees have a leg up on the other two options as far as quickly sorting as it is the fastest at that, but the searching speed depends on the height of the tree itself.

I believe that vectors are the best option, as the quick sort option is extremely helpful and the search is faster than the binary tree option.