Alexander Dollison

SNHU

CS-300

12/01/2023

Project One: Pseudocode and Runtime Analysis

Vector Data Structure

Opens Files

Create function with two parameters

- Open file

- read data

- parse each line

- check for course title

- check for course number

IF the file is free of errors,

check to see if prereq exists in course file

CreateCourseObj (create parameters)

{

intialize variables for courses, and read file;

open the file to read again

WHILE file is open

store the course obj in a vector data structure

}

SearchforSpecificCourse ()

{

initialize variables for opening file

open file

WHILE file is open

print course information

store data gathered in a data structure

}

Hash Table Pseudocode

Start Program

- Open file

- read data

- parse each line

- check for course title

- check for course number

- IF prereq is found

- add to array

- IF course parameters < than two

- skip course

- display error msg: File isn't formatted Properly

- end program

- ELSE

- add course name, course number, & prereq to hash table

IF prereq exists

- check to see if prereq comes before the course

- add to has table

IF prereq is not found

- skip course

- display error msg

Create function with parameters

intialize variables for courses, and read file;

open the file to read again

WHILE file is open

store the course obj in a vector data structure

SearchforSpecificCourse ()

{

initialize variables for opening file

open file

WHILE file is open

print course information

store data gathered in a data structure

}

CS-300: Milestone Tree Data Structure Pseudocode

1.

Start program

create two nodes variables for left and right;

create root variable set it to null;

create variables for course name, and an integer for course number;

open file

WHILE file is open

read data

parse each line

- check for course title

- check for course number

IF root != null

check if a prereq found

add prereq to right node

IF course parameters are < two

add course to left node

display error msg

ELSE

add course name, num, and preq to right node

- Display Results

- close file

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

Start Program

create variables for course objects;

create root variable and set it to null;

create left and right node variables;

Open file

WHILE file is open

read file

parse each line

IF root == null

SEARCH for course

IF course is found

create course object;

ADD course to right node

IF course is not found

print error msg

close file

stop program

close file

print objects

end program

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

create root variable and set it to null;

create left and right node variables;

open file

WHILE file is open

read file

parse each line

IF root == null

- check for course title

- check for course number

IF root == null

check if a prereq found

add prereq to right node

IF course parameters are < two

add course to left node

display error msg

ELSE

add course name, num, and preq to right node

display msg

close file

end program

4. Pseudocode for a Menu

create an integer for switch statement, set it to 0 name it uInput;

WHILE input does not equal 4;

PRINT 1. Load Data Structure

PRINT 2. Course List;

PRINT 3. Course;

PRINT 4. Exit

SWITCH (uInput):

case 1:

loadBids(bid);

break;

case 2:

Print: Course List;

break;

case 3:

Print Course;

break;

case 4:

Print Message: "Thank you";

end program

break;

Default:

PRINT: no input found from user;

break;

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

create a sorting string with the parameter string s

create a char that sets the length +1

create a string to character array

then sort the array

create two integers for alphabet, and numbers

create a while loop for if the alphabet integer is less than 97

set the alphabet int to +1

create a for loop,

IF i is less than 97 then set the number int to +1

ELSE set the alpha int to +1

and return s (outside of the for loop)

DRIVER

create a string for the classes, and print the courses in alphanumeric order

5. When we consider the advantages of each type of data structure, we note that creating a vector is one-dimensional, which can be used to handle the data for this specific assignment. We are only handling courses and reading files, so there is no reason to use anything more complicated than vectors to assign value to the data in these files. It will use less memory than other data structures, however the elements will not be capable of deletion, and the main file would not contain multiple data types.

A hash table will allow information to be organized, stored, and referenced later through a project with a key. This is greatly advantageous for the project since we can create, delete, and call variables within the project, we can create unique elements and synchronize these elements throughout the table. Hash tables do, however, affect the speed of the program because it must synchronize every time it is run.

The creation of a tree would allow better organization, such as data that can be stored in the left or right branches beneath the first variable and expanded accordingly. We can run searches through the program and find specific variables faster than with other methods, but like hash tables, it takes a while to modify the tree once it has been created.

6. Of the three types of data structures, I would recommend using Hash tables. While it may have a longer runtime than vectors, careful planning and its ease of access can eliminate unnecessarily long runtimes. It is more organized and should read and compare files more accurately than with vectors or trees. Aside from the base functionality of sorting courses alphabetically, hash tables leave room to add additional functionality to the program should it need refining, and additional courses can be added at any time.

Trees linked below and labeled.

VECTOR

|  |  |  |  |
| --- | --- | --- | --- |
| Code | Line Cost | # Times Executed | Total Cost |
| For all courses | 1 | n | n |
| If the course is the same as courseNumber | 1 | n | n |
| Print out the course information | 2 | 1 | 1 |
| For each prereq of course | 1 | n | n |
| Print the prereq of course info | 2 | n | n |
|  |  | Total Cost | 6n+1 |
|  |  | Runtime | 1(n) |

Hash Table

|  |  |  |  |
| --- | --- | --- | --- |
| Code | Line Cost | # Times Executed | Total Cost |
| For all courses | 2 | n | n |
| If the course is the same as courseNumber | 1 | n | n |
| Print out the course information | 1 | 1 | 1 |
| For each prereq of course | 2 | n | n |
| Print the prereq of course info | 4 | n | n |
|  |  | Total Cost | 9n+1 |
|  |  | Runtime | 0(n) |

Tree

|  |  |  |  |
| --- | --- | --- | --- |
| Code | Line Cost | # Times Executed | Total Cost |
| For all courses | 1 | n | n |
| If the course is the same as courseNumber | 1 | n | n |
| Print out the course information | 2 | 1 | 1 |
| For each prereq of course | 1 | n | n |
| Print the prereq of course info | 4 | n | n |
|  |  | Total Cost | 8n+1 |
|  |  | Runtime | 0(n) |