Ashley De Venuto

Instructor Atkinson

CS-300: Analysis and Design

10/08/2022

Project 1

* **//**vector Data Structure
* Read File
* Use fstream to open file
* Create void method loadCourses(string, dataStructure)
* Call to open file, if return value = ‘-1’, file not found
* ELSE file not found
  + WHILE != End of File
    - IF < 2 values per line return ERROR
    - ELSE read parameters
    - IF > 2 parameters
      * IF parameter over 2 is first elsewhere, continue
      * ELSE return ERROR
* Close file

**Hash Table Pseudocode**

* Initialize Course Vector nodes
* Create Class for HashTable
* Create Insert method
* Create while and for loop through file
* While != end of file
  + For every line within file
  + For 1st and 2nd value
    - Create temp item
  + If 3rd value exists
    - Add to current value
  + Call insert method

Search and Print from HashTable

* User puts in input
* Input is assigned to key
* If key found
  + Print course information
  + For prerequisite of course(s)
    - Print prerequisite course information

**Tree Data Structure Pseudocode**

* Define class Binary Tree
* Create root->null
* Create Insert Method
  + IF root = null current course = root
  + ELSE IF course number < root, add to left side
    - IF left = null add course number
    - ELSE
      * IF course number is < leaf add to left side
      * IF course number is > leaf add to right side
  + ELSE IF course number < root, add to right side
    - IF right = null add course number
    - ELSE
      * IF course number is < leaf add to left side
      * IF course number is > leaf add to right side

**Search and Print from Tree:**

* Get user input
* Create Print Method
  + IF root != null
    - Transverse left output if found
    - Transverse right output if found

**Design pseudocode for creating course objects and store in appropriate data structure**

* Start program
* Create parameter for course objects
* Set root variable to null
* Create node variables for left and right sides
  + Open file
  + WHILE file is open
    - Read file
    - Parse every line
* IF root == null
  + Search for course
  + IF course found
    - Create course object
    - ADD course to right node
* IF course not found
  + Print ERROR
  + Close file
  + Stop program
* Close file
* Print objects
* End program

**Design pseudocode that prints course information and prerequisites**

* Create root and set to null
* Create nodes for left and right side
  + Open file
  + WHILE file is open
    - Read file
    - Parse lines
    - IF root == null
      * Check for title of course
      * Check for number of course
    - IF root == null
      * Check if prerequisites are found
      * ADD prerequisites to right node
    - IF course parameters = < 2
      * ADD to left side
      * Display ERROR
  + ELSE
    - ADD name, number, and prerequisite of course to right node
  + Display message
  + Close file
  + End program

**Pseudocode for Menu**

* Create and set integer for switch statement to 0 and name it userInput
* Create Bid variable to access menu
  + WHILE input != 4;
  + PRINT 1. Load Data Structure
  + PRINT 2. Print Course List
  + PRINT 3. Print Course
  + PRINT 4. Exit
    - SWITCH(userInput)
      * Case1:
        + loadBid(bid);
        + break;
      * case2:
        + print(Course List)
        + break
      * case3:
        + print(Course)
        + break;
      * case4:
        + print(“Exit”)
        + end program
        + break;
      * Default:
        + Print(“no input found from user”)

**Create pseudocode that list the courses of computer science in alphabetical order**

* Create sorting string
* Create char that sets length to 1
* Create string for character array
* Sort array
* Create a variable alphabet and numbers
* Create WHILE loop for alphabet integer is lesser then 97
* Set alphabet to +1
* Create for loop
* If variable < 97 set number to set +1
* Else set alpha to +1
* Return
* Create a string for the classes, print courses in alphabetical order.

Vector

|  |  |  |  |
| --- | --- | --- | --- |
| Code | Line Cost | Times Executed | Total Cost |
| For all courses | 1 | n | n |
| If the course = courseNumber | 1 | n | n |
| Print course info | 2 | 1 | 1 |
| For each prerequisite of the course | 1 | n | n |
| Print prerequisite course info | 2 | n | n |
|  |  | Total Cost | 7n +1 |
|  |  | Run Time | O(n) |

Hash Table

|  |  |  |  |
| --- | --- | --- | --- |
| Code | Line Cost | Times Executed | Total Cost |
| For all courses | 2 | n | n |
| If course = courseNumbers | 1 | n | n |
| Print course info | 1 | 1 | 1 |
| For each prerequisite of course | 2 | n | n |
| Print prerequisite course info | 4 | n | n |
|  |  | Total Cost | 10n - 1 |
|  |  | Run Time | O(n) |

Tree

|  |  |  |  |
| --- | --- | --- | --- |
| Code | Line Cost | Times Executed | Total Cost |
| For all courses | 1 | n | n |
| If the course = courseNumber | 1 | n | n |
| Print course info | 2 | 1 | 1 |
| For each prerequisite of the course | 1 | n | n |
| Print prerequisite course info | 4 | n | n |
|  |  | Total Cost | 9n +1 |
|  |  | Run Time | O(n) |

The Vector’s disadvantage is the fact that it must go through the whole list to find a specific course until the match is found. The advantage of a vector is speed. Vectors can read files the fasted and can add course objects. They also have the best run time out of the three. Hash tables can search a list the fastest out of the three data structures. They have slower implementation time when creating a list. In order to print the list, the courses value must be extracted and sorted. Due to these factors, this data structure should not be used. The Binary tree beats the vector in speed for sorting. I would recommend using a vector sort for the project. Being able to sort a whole catalogue of values quickly is better in the long run for this project. The overall usability of the data structure is more important than being slow at searching a list. Therefore, I believe that the vector sort is the best option.