CS300

Dr. Webb

Julia Coronado

February 25, 2024

**Pseudocode and Runtime Analysis**

**Vector Pseudocode**

Open file

* *Using if file stream*
* Read data in file.
* Read lines in file.

IF

* File contains error return null.

IF

* No errors continue to check for course name.
* Course title
* Course number

Create object for course.

* Parameters

Variables for courses

* Read file.
* *Ifstream MyReadfile*

WHILE file open

* Course obj store data in vector
* *Vector<Course> course*

Search For Course

* Open file

IF File is open

* Read data in file.
* *Std::fstream*
* Check for course information.
* Print desired course information.
* *Cout<< course<<*
* Store in vector.

**Hash Table Pseudocode**

Open File

* Read data in file.
* *fstream*
* Parse Lines

IF No errors continue to open file

WHILE file is open

* Check course title.
* Course name

IF Course Parameters do not match

* Return error message.
* *Return null*
* End

ELSE if course found.

* Add to hash table array.

Search for a course.

IF file is open

* Search for a course.
* *Cin >> Course;*
* Read lines in file.
* Print out course information.
* Cout << *courseinformation*
* Store course data in hash table array.

**Binary Search Tree Pseudocode**

Initialize

Create Nodes

* *Node right=left=nullptr*
* Create Root node = null
* Create the right node.
* *Node\*right*
* Create the left node.
* *Node\*left*

Create Course Variable

* Course name
* *String name*
* Course number
* *String course number*

Opening file

* Open file
* *Load data from file*
* Read data in file.
* *Process lines*
* Parse lines.

IF file is open

* Check course name.
* Course number

IF course has pre req

* Add to left node.
* Course has no pre req add to right node.
* *Cout << prerequisites*

Create Course Objects

* Initialize variables created for course information.
* *Recursive node (course,root)*
* Create root.
* Create Nodes

IF file is open

* Read data in file.
* Parse lines.

Root= null

IF course found

* Object
* Store in right node.
* *Course Node right*

IF course not found

* Return error message.
* Close application

Search for a course.

Initialize course variables.

* Create root node.
* Create left right nodes.

Open File

IF file is open

* Read data in file.
* *getline*
* Parse lines.
* *String line*
* Check course name.
* Check course number.

IF course found

* Add to right node.

IF course not found

* Display error message.
* Close application

ELSE

* Add course information into the right node.
* *Insert recursive(course,node)*
* Print out course information.
* Close application

**Menu Pseudocode**

* *Cout << following menu options<<*

*Choice* {

Print 1 Load Course

Print 2 Print sorted course.

Print 3 Course Information

Print 4 Exit

*Choice ==1 load course information*

*Choice ==2 cout<< printlist<<*

*Choice ==3 cin >>coursename*

*Choice ==4 cout << goodbye*

**Pseudocode for Alphanumeric Order**

**Vector Sorting**

* Open file
* Sort data in vector file alphanumerically.
* Display list.
* Sort Course data.
* *Sort courses()*
* Display course data.
* *Print courses()*

**Hash table Sorting.**

* Open file
* *Course list*
* Print course data.
* Sort data in hash table array.
* Display Course data.

**Binary Tree Sorting**

* *Load from file*
* Open file
* Root node
* Sort data in right node.
* Display course information in the right node.

**Runtime Analysis**

|  |  |  |  |
| --- | --- | --- | --- |
| **Vector Sorting** | **Hash table Sorting** | **Binary Tree Sorting** |  |
| Creating (1) | Creating (1) | Creating (1) |  |
| Reading line(n) | New Line (1) | root (n) |  |
| Vector Creation(n) | New node(n) | Left node (n) |  |
| Recommendation (n) | Add node (n) | Right node(n) |  |
| adding(n) | append node(n) | Lines in file (n) |  |
| changing(n) | assign key(n) | Append (n) |  |
| return null(n) | next node(n) | prerequisite (n) |  |
| **Total Line Cost= 6n** | insert (n) | append prerequisite(n) |  |
| **O(n)** | Return null(n) | Insert (n) |  |
| **Total = 6n+1** | Set key to new node(n) | **Total line cost= 10n** |  |
|  | Pre req(n) | **O(n)** |  |
|  | **Total Line cost = 12n** | **Total =10n+1** |  |
|  | **O(n)** |  |  |
|  | **Total 11n+2** |  |  |

**Advantages and Disadvantages**

**Hash Table Sorting**

Advantages:

* Good to use for larger databases.
* Use of the keys makes it easier to access information.
* Fast information return with queries

Disadvantages:

* It uses a bit more memory than the other sorting methods.

**Vector Sorting**

Advantages:

* Easy to use.
* Faster indexing.
* Lower memory usage

Disadvantages

* The insertion order can make for slower search results.

**Binary Tree Sorting**

Advantages

* Faster search results
* Better Organization

Disadvantages

* More memory usage
* The modification process is longer.

**Conclusion**

After researching all these different methods and writing pseudocode for each I think I know what the best approach is for the data that we are using. I think I would use the vector sorting method because it is quicker and uses less memory. Although it is a basic implementation it is easy to use and based on the data will allow for faster indexing of information. Hopefully ABCU considers my recommendation as I believe this is the best approach for moving forward.