Saram Nadeem

CS – 300

10/8/23

Project One

Linked List:

**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 label for course number, title, and prerequisite

Open course information file

If the file is not found return an error

Else, create a tree for all of the courses

Check for an existing line in the file and read it

If a line has more than two connections return an error

Otherwise return course information to respective labels

Read next line in file

If course number is not a number, end and return error

Move to next line

If course title does not have letters return error

Move to next line

If course prereq does not have letters return an error

Move to next line

If none of them are linked in courses then return error

Connect courses to children and parent nodes

Return

End

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

Create course and store information in data structures tree

Create main course class

Define course number

Define course title

Define course prereq

Read file into and return to course number (this.courseNum = courseNum)

Read file info and return to course title (same as above)

Read file info and return to course prereq (same as above)

END

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

Create a function that shows course info and can move around list

Function shows number, title, and needed prereq

If a course is selected

Show children and parent relationships needed

Return function to choose courses again

End

|  |  |  |  |
| --- | --- | --- | --- |
|  | Cost per line of code | Number of executions | Cost |
| All courses | 1 | N | N |
| Course number | 1 | N | N |
| Course info | 2 | 1 | 1 |
| Prerequisite | 1 | N | N |
| Print info | 2 | N | N |
| Total Cost |  |  | 6n+1 |
| Total Runtime |  | 1(n) |  |

Hash Table:

**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.**

**Open the given file**

**If the file is not found return an error**

**Else if it is found then continue**

**While the last line has not been read**

**Read all of the lines**

**If there are less than two parameters per line then return an error**

**Otherwise continue reading the lines**

**If there are three parameters and if the third parameter is found again in the list then continue**

**Otherwise print an error**

**Close file and finish program**

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

Initialize a class for the hash table

Implement a method to insert items into hash table

Read through the file

While the last line has not been read

Create a temp value to store first and second values

If there’s specifically three values

Then insert that to the current value

And return the method for each value

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

Ask to receive an input

Create a key and assign the input to it

If the key is not found return an error

If it is found

Return the information for each course

Return the prerequisite for each course as well

|  |  |  |  |
| --- | --- | --- | --- |
|  | Cost per line of code | Number of executions | Cost |
| All courses | 2 | N | N |
| Course number | 1 | N | N |
| Course info | 1 | 1 | 1 |
| Prerequisite | 2 | N | N |
| Print info | 2 | N | N |
| Total Cost |  |  | 8n + 1 |
| Total Runtime |  | O(n) |  |

Tree Table:

**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.**

Open the file

Read the data within the file

If there are less than two parameters on each line

Return error

Else read the parameters

If there is a third parameter on the line

Then check to see if that parameter matches elsewhere in the file

Else return an error

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

**Create a class for the course objects**

**Read through the opened file (process it)**

**For each line that is read**

**Insert the course name and ID**

**When a third value is found by the last code**

**Insert that parameter or required prereq**

**Create a binary tree class for the data structure**

**Create the root of the tree**

**Set that equal to null**

**Create a way to insert the data into the tree**

**If the tree root is null, then the read course will be the root**

**If the course number is not equal or less than our root**

**Then add to the left side**

**If the found course number is greater then the compared leaf**

**Add to the left**

**If the found course number is less then the compared leaf**

**Add to the right**

**If the found course number is greater than our current root value**

**Add to the right**

**Else add to the left**

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

Receive the data through an input

Create a function to print it out

If our current root is not null

Move through the tree and return either left or right side

|  |  |  |  |
| --- | --- | --- | --- |
|  | Cost per line of code | Number of executions | Cost |
| All courses | 1 | n | N |
| Course number | 1 | N | N |
| Course info | 2 | 1 | 1 |
| Prerequisite | 1 | N | N |
| Print info | 3 | N | N |
| Total Cost |  |  | 8n + 1 |
| Total Runtime |  |  | O( log(n)) |

Advantages and disadvantages

There are quite a few advantages when it comes to vectors. The biggest two in our application would be how simple they are to implement and use, and their efficiency memory wise. The biggest disadvantage that can be recognized would be how slow it can be to delete or insert elements where you want them. If there is going to be constant change, I wouldn’t recommend this one.

Hash tables have a great deal of advantages. These would include speed when looking to insert or delete operations, and dynamic sizing which would allow the table to resize automatically. For their disadvantages, they do require more memory just to get started, but their resize ability is useful because it can optimize its memory usage.

When using tree tables, their advantages allow for quick and efficient use if there is a lot of data that’s being processed. One of the biggest disadvantages there would be that if a tree is unbalanced, it would lead to larger loading times which would lead to less overall efficiency.

Recommendation

If I had to recommend a single method, it would have to be the hash table. This method is useful for operation as it can provide fast performance when looking to insert or delete operations. It’s ability to resize and prioritize memory usage can be extremely useful If the data size increases, so it’s more than suitable for our scenario. If the data size ever needs to increase, a transition into a tree table can be implemented.