Project One

Callie Oleson

4/9/2023

**LinkedList PrintAll(couseInfo, lowIndex, highIndex)**

Set midpoint = lowIndex + (highIndex – lowIndex) / 2

Set pivot = vector<courseInfo>[midpoint]

Set done = false

While not done:

While lowIndex is less than pivot

Increment low index

While pivot is less than highIndex

Decrement high index

If lowIndex is greater than or equal to highIndex

Set done to true

Else

Set temp node to lowIndex

Set lowIndex to highIndex

Set highIndex to temp node

Increment lowIndex

Decrement highIndex

Output vector<courseInfo> to console

**Hash PrintAll()**

Course course

Unsigned key = hash atoi course cstring

For all courses:

If key is not equal to max

Output courseInfo to console

courseNode is equal to next node

while courseNode is not null

output courseInfo to console

courseNode is equal to next node

**Tree PrintAll(currentNode)**

If currentNode is null

Return

PrintAll(currentNode->left)

Print currentNode

PrintAll(currentNode->right)

**Menu:**

While userInput is less than 4

Output “Course List Menu:”

Output “1: Load Course List”

Output “2: Print Course List”

Output “3: Print Course and Prerequisites”

Output “4: Exit Program”

Get input from user

Switch(userInput)

Case 1:

loadCourseInfo(courseInfo)

output “Course List Loaded”

break

Case 2:

printAll()

break

Case 3:

Search(courseInfo, courseInfo.size, userInput)

Break

**LinkedList Runtime Analysis:**

|  |  |  |  |
| --- | --- | --- | --- |
| For all lines in file | 1 | N | N |
| If line has 2 items | 1 | N | N |
| Course.courseId = firstItem | 1 | 2 | 2 |
| Course.courseName = secondItem | 1 | 2 | 2 |
| Go to next line | 1 | 3 | 3 |
| If line has more then 2 items | 1 | N | N |
| Check 3rd and 4th items to see if match can be found | 1 | 1 | 1 |
| If not match found | 1 | N | N |
| If match found | 1 | N | N |
| Course.coursePrereq1 = third item | 1 | 1 | 1 |
| Course.coursePrereq2 = fourth item | 1 | 1 | 1 |
| For entire vector<courseInfo> | 1 | N | N |
| If course matches userInput | 1 | N | N |
| Print course to console | 1 | 1 | 1 |
| If course contains prerequisite | 1 | N | N |
| Print prerequisite info to console | 1 | 1 | 1 |
| Print “Enter course name or ID number | 1 | 1 | 1 |
| Get userInput | 1 | 1 | 1 |
| Search(courseInfo, courseInfo.size, userInput) | 7 | 1 | 7 |
|  |  |  |  |
|  |  |  |  |
| **Total Cost** |  |  | 21+8N |
|  |  |  | O(N) |

**Hash Table Runtime Analysis:**

|  |  |  |  |
| --- | --- | --- | --- |
| For all lines in file | 1 | N | N |
| If line has 2 items | 1 | N | N |
| Courseinfo = line | 1 | 2 | 2 |
| Insert courseInfo into courses | 1 | 2 | 2 |
| Go to next line | 1 | 3 | 3 |
| If line has more than 2 items | 1 | N | N |
| Check if 3rd or 4th item has a match | 1 | 1 | 1 |
| If match not found | 1 | N | N |
| else | 1 | N | N |
| For all courses | 1 | N | N |
| If first item of string is equal to courseID | 1 | N | N |
| Print courseInfo | 1 | 1 | 1 |
| For prerequisities | 1 | 1 | 1 |
| Print prerequisite courseInfo | 1 | 1 | 1 |
|  |  |  |  |
| **Total Cost** |  |  | 11 + 7N |
|  |  |  | O(N) |

**Tree Runtime Analysis:**

|  |  |  |  |
| --- | --- | --- | --- |
| set root to null | 1 | 1 | 1 |
| if root is null | 1 | N | N |
| Root is equal to new node course | 1 | 1 | 1 |
| Else | 5 | N | 5n |
| add node root and course | 1 | 1 | 1 |
| if node is larger | 1 | N | N |
| Add to left | 1 | 1 | 1 |
| If no left node | 1 | N | N |
| node becomes left | 1 | 1 | 1 |
| Recurse down left node | 1 | 1 | 1 |
| If no right node | 1 | N | N |
| Recurse down right node | 1 | 1 | 1 |
| Initialize path to file | 1 | 1 | 1 |
| For all lines in file | 1 | N | N |
| If line has 2 items | 1 | N | N |
| Course.courseId = first item | 1 | 2 | 2 |
| Course.courseId = second item | 1 | 2 | 2 |
| If line has more than 2 items | 1 | N | N |
| Check 3rd and 4th item for match | 1 | 1 | 1 |
| If match not found | 1 | 1 | 1 |
| Go to next line | 1 | 3 | 3 |
| If match found | 1 | N | N |
| Course.coursePrereq1 = 3rd item | 1 | 1 | 1 |
| Course.coursePrereq2 = 4th item | 1 | 1 | 1 |
| Insert node to tree | 1 | 2 | 2 |
| Set temp node equal to root | 1 | 1 | 1 |
| While current node not null | 1 | N | N |
| If match found | 1 | N | N |
| Output course to console | 1 | 1 | 1 |
| For all prerequisites | 1 | N | N |
| Output prerequisites to console | 1 | 1 | 1 |
| If courseNumber smaller than currentNode | 1 | N | N |
| Set current node to left node | 1 | 1 | 1 |
| Set current node to right node | 1 | 1 | 1 |
|  |  |  |  |
| **Total Cost** |  |  | 26 + 17N |
|  |  |  | O(N) |

All three data structures have their advantages and disadvantages. When it comes to the linked list, it makes it pretty simple to parse through the list while searching for a specific course and sorting the list to print in order, despite being relatively simple and a bit crude and bulky. The hash table on the other hand, uses less code to do the same thing, but is better suited to organizing items with a single data point rather than items with multiple points per item. The binary tree uses a lot of code for each data point and is extremely bulky, but organizes the data really well and it is able to quickly and effectively search for a course and organize the course list. In my opinion, the binary tree would be the best way to organize the data in a program, even though the runtimes for each data structure don’t differ much.