6-2 Submit Project One

Matthew Taylor

02/18/2024

Menu Pseudocode:

Int choice = 0

While choice != 9 {

Print menu

Print 1. Load courses

Print 2. Display all courses

Print 3. Find course

Print 9. Exit

Print enter choice

Cin choice

If choice isn’t 1-3 or 9 then throw an error

Case 1 parseText(filename)

Case 2 printAll()

Case 3 displayCourse(search(courseId)

Case 9 print goodbye and exit application

}

//vector sort and print function

quickSort(vector, lowIndex, highIndex)

if (lowIndex < highIndex)

pivotIndex = partition(vector,lowIndex, highIndex)

quickSort(vector, lowIndex, pivotIndex - 1)

quickSort(vector, pivotIndex, highIndex)

partition(vector, lowIndex, highIndex) {

midpoint = lowIndex + (highIndex - lowIndex) / 2

pivot = vector[midpoint].CourseId

done = false

while not done{

Increment lowIndex while vector[lowIndex].courseId < pivot

Decrement highIndex while pivot < vector[highIndex].courseId

If zero or one elements remain

done = true

Else

Swap vector[lowIndex] and vector[highIndex]

Increment lowIndex

Decrement highIndex

}

Return highIndex

}

printAll(){

quicksort(vector, 0, vector size -1)

for each course in vector

displayCourse(course)

}

//for the hash table, I don’t believe that a hash table itself can be sorted. I could read the entire table into a different data structure and then sort it.

//for the binary search tree, the InOrder method already defined in the BinarySearchTree class will print the list of courses in the correct order.

//vector analysis

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| create a vector called Courses that holds objects of type Course | 1 | 1 | 1 |
| open the file specified by filename | 1 | 1 | 1 |
| loop through each line of the file | 1 | n | n |
| if the line has at least two arguments | 1 | n | n |
| create a variable called course of type Course | 1 | n | n |
| course courseId property equals the first argument | 1 | n | N |
| course courseName property equals the second argument | 1 | N | N |
| add remaining arguments to the course property coursePrerequisiste | 1 | N | N |
| add course to the vector Courses | 1 | N | n |
| return Courses | 1 | 1 | 1 |
| ValidateCourselist() | 4n+2 | 1 | 4n+2 |
| **Total Cost** | | | 11n + 5 |
| **Runtime** | | | O(n) |

//Hash Table analysis

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| open the file specified by filename | 1 | 1 | 1 |
| loop through each line of the file | 1 | n | n |
| if the line has less than two values | 1 | n | n |
| print line to debug console with an error message | 1 | n | n |
| else | 1 | n | n |
| create a variable called course of type Course | 1 | n | n |
| course courseId property equals the first argument | 1 | n | n |
| course courseName property equals the second argument | 1 | n | n |
| if there are more than two arguments | 1 | n | n |
| loop through remaining arguments | 1 | n | n |
| add argument to the coursePrerequisite vector | 1 | n | n |
| Insert(course) | 12 | 1 | 12 |
| validateCourseList() | 10n | 1 | 10n |
| return nodes | 1 | 1 | 1 |
| **Total Cost** | | | 20n + 14 |
| **Runtime** | | | O(n) |

//binary search tree analysis

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| open the file specified by filename | 1 | 1 | 1 |
| loop through each line of the file | 1 | n | n |
| if the line has less than two values | 1 | n | n |
| print line to debug console with an error message | 1 | n | n |
| else | 1 | n | n |
| create a variable called course of type Course | 1 | n | n |
| course courseId property equals the first argument | 1 | n | n |
| course courseName property equals the second argument | 1 | n | n |
| if there are more than two arguments | 1 | n | n |
| loop through remaining arguments | 1 | n | n |
| add argument to the coursePrerequisite vector | 1 | n | n |
| Insert(course) | 10 | 1 | 10 |
| validateCourseList(root) | 7n | 1 | 7n |
| return root | 1 | 1 | 1 |
| **Total Cost** | | | 17n + 12 |
| **Runtime** | | | O(n) |

All three data structures have advantages and disadvantages. One of the advantages of a vector is that adding data to the vector is fairly quick. However, when it comes to searching or sorting the vector, it will end up being one of the slower options. Hash tables are much faster than vectors when it comes to searching, but when it comes to adding new values to the table, it can take a bit longer due to collisions and the hashing that needs to happen first. The hash table is also really terrible to sort. It must be converted to a different data structure before it can be sorted which takes a lot more time and memory. The binary search tree does take a bit longer to add new values than the vector, but when it comes to searching and sorting, it is really efficient, especially considering the fact that the tree can be traversed in order and is essentially already sorted.

I would recommend using the binary search tree because the requirements detail that the data may need to be sorted and searched frequently. As far as the runtime goes for loading the data into the data structure, It may be slightly slower, but the runtime big O results are the same for each of the data structures. However, when it comes to sorting, the binary search tree is significantly better as it is already sorted and simply requires traversing the tree in order.