Mao Christie

June 13, 2024

CS-300

Project 1

**PSEUDOCODE:**

Menu**:**

* Initialize data structure
* Initialize a variable called userChoice = 0
* While userChoice is not equal to 9
  + Print “Option 1. Load the file”
  + Print “Option 2. Print an ordered list of all courses”
  + Print “Option 3. Print the course title and prerequisites of a course”
  + Print “Option 9. Exit”
  + Get userChoice = user input
  + Switch (userChoice)
    - Case 1
      * Print “Loading data…”
      * Load data to data structure using the file through loadData method
      * Print “Finished”
      * break
    - Case 2
      * Use sortedCourses method, passing the loaded data structure
      * Use sortedPrint method, passing sorted courses data structure
      * Break
    - Case 3
      * Initialize courseNum
      * Print “What course are you looking for?”
      * Get courseNum from the user
      * Pass courseNum to searchCourse method
      * Break
    - Case 9
      * Print “Goodbye”
      * Break
    - Default
      * Print “That is not an option”
      * Break

Sort and print Pseudocode:

Void sortedInsert(data structure headNode, newNode) {

* Initialize currentNode
* If headNode is null
  + headNode = newNode
* Else if headNode’s course # is greater than or equal to newNode’s course #
  + The node next to newNode (newNode -> next/right) = headNode
  + newNode’s next’s prev (newNode->next->prev) = newNode
  + headNode = newNode
* else
  + currentNode is equal to headNode
  + while currentNode’s next (currentNode->next/right) is not null AND currentNode’s next course # is less than newNode’s course #)
    - currentNode is equal to the node to the right(next) of currentNode
  + newNode’s next is equal to currentNode’s next
  + if the node next to currentNode is not null
    - newNode-next->prev = newNode
  + the node next to currentNode is equal to newNode
  + the node prior to newNode is equal to currentNode

}

void sortedCourses(data structure Node head){

* Initialize “sorted” = null
* Initialize currentNode = head
* While currentNode is not null
  + Initialize next = node next to currentNode (currentNode->next/right)
  + Current->prev = current->next = null
  + Use sortedInsert method, passing “sorted” and currentNode
  + currentNode = next
* head = sorted

}

Void sortedPrint(data structure courses)

* Initialize i
* For all courses
  + Print out the course title and number

**RUNTIME ANALYSIS:**

Vector:

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **for all courses** | 1 | n | n |
| **if the course is the same as courseNumber** | 1 | n | n |
| **for each prerequisite of the course** | 1 | 1 | 1 |
| **for each prerequisite of the course** | 1 | n | n |
| **print the prerequisite course information** | 1 | n | n |
| **Total Cost** | | | 4n + 1 |
| **Runtime** | | | O(n) |

|  |  |  |  |
| --- | --- | --- | --- |
| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| Initialize an empty vector called Courses | 1 | 1 | 1 |
| Initialize string variable called fileLine | 1 | 1 | 1 |
| Initialize list | 1 | 1 | 1 |
| Open a text file called course information | 1 | 1 | 1 |
| If the file opens | 1 | 1 | 1 |
| While the file has a next line | 1 | n | n |
| fileLine = the line in the file | 1 | n | n |
| Split the line by “,” and add into list | 1 | n | n |
| If the list has at least 2 parameters | 1 | n | n |
| Create a new course object | 1 | n | n |
| If the list has more than 2 parameters | 1 | n | n |
| For each token after index 1 | 1 | 1 | 1 |
| For each token after index 1 | 1 | n | n |
| Add parameter to course with addPrerequisite function | n | n | n |
| Add course to vector with pushback | 1 | n | n |
|  |  | Total cost | 9n + 6 |
|  |  | runtime | O(n) |

Hash Table:

|  |  |  |  |
| --- | --- | --- | --- |
| **Code** | **Line Cost** | **# times executes** | **Total cost** |
| Initialize an empty hash table called courseTable | 1 | 1 | 1 |
| Intialize the CSV parser using the given path | 1 | 1 | 1 |
| Open a file called course information | 1 | 1 | 1 |
| If the file opens | 1 | 1 | 1 |
| While the file still has a next line | 1 | n | n |
| Initialize an empty list called list | 1 | n | n |
| fileLine = the line in the file | 1 | n | n |
| Split the line by “,” and add to the list | 1 | n | n |
| If the list has at least 2 items | 1 | n | n |
| Create a course object with course# and name | 1 | n | n |
| If the list has more than 2 items | 1 | n | n |
| If the item exists in the hash table | 1 | n | n |
| Add the parameter to the course object | 1 | n | n |
| Insert course object to the hash | 1 | n | n |
| Go to the next line | 1 | n | n |
|  |  | **Total Cost** | **11n + 4** |
|  |  | **Runtime** | **O(n)** |

Tree:

|  |  |  |  |
| --- | --- | --- | --- |
| Code | Cost | # times executes | Total cost |
| Initialize an emoty binary search tree called bst | 1 | 1 | 1 |
| Initialize the CSV parser with path | 1 | 1 | 1 |
| Open a file called “Course Information” | 1 | 1 | 1 |
| If the file opens | 1 | 1 | 1 |
| While the file has a next line | 1 | n | n |
| Open a file called “Course Information” | 1 | n | n |
| Initialize fileLine = the current line in the file | 1 | n | n |
| Split the line by “,” and add each split text to the list | 1 | n | n |
| If the list has at least 2 items(parameters) | 1 | n | n |
| Create a course object with parameters | 1 | n | n |
| Insert course object into bst | 1 | n | n |
| Go to the next line | 1 | n | n |
|  |  | Total cost | 8n +4 |
|  |  | runtime | O(n) |

Evaluation:

Based on the advisor’s requirements, all three data structures have their advantages and disadvantages. A vector allows for constant-time access to the elements contained within the vector by index. Inserting or deleting elements can be inefficient compared to the other data structures, however. A hash table can (depending on the hashing functions) be very fast. The worst case time is O(n) but a hash table without collisions, etc. can have O(1) time complexity. A disadvantage, however, is that if the number of courses grow bigger than the available space in the hash table, a new table must be created and all the courses must be rehashed. A binary search tree is more memory efficient and can store an arbitrary amount of data. A disadvantage is that operations (inserting, deleting, and searching within the tree) are dependent on the height of the tree. I would recommend the binary search tree based on its more efficient memory.