Ivan Arroyo

October 3, 2024

Professor Lusby

CS-300

Project One

**Vector Pseudocode**

LoadCourseData(fileName):

Attempt to open the file using the (fileName)

If the file cannot be opened, display the error message and end

While more data remains in the file:

Read data within the file

Split the data using commas and place it into the list (courseData)

If the length of courseData < 2 (missing course number or title)

Display an error message indicating the missing data, then end

For each prerequisite in courseData:

If the prerequisite is not listed as a course in the file

Display the error indicating the prerequisite is missing, then end

Call createCourseObject using courseData as the input

Close the file and end

createCourseObject (courseData):

Create a new course object

Set courseNumber to the first value of courseData

Set course title to the second value of courseData

If any prerequisites after the third value remain

For each prerequisite remaining in courseData:

Add this prerequisite to the course object's list

Add this course object to coursesVector (holds all courses)

End

SearchCourse (coursesVector, courseNumber):

For each course in the coursesVector

If the courseNumber == the input courseNumber

Display the course's title

If the course has any prerequisites

Display "Prerequisites:"

For each prerequisite in the course's list of prerequisites:

Display the prerequisite

If no prerequisites exist, display "No prerequisites"

Stop the search

If no course with courseNumber is found, display "Course not found."

End

**Hash Table Pseudocode**

LoadCourseData(fileName):

Attempt to open the file using the (fileName)

If the file cannot be opened, display the error message and end

Initialize a hash table

While more data remains in the file:

Read data within the file

Split the data using commas and place it into the list (courseData)

If the length of courseData < 2 (missing course number or title)

Display an error message indicating the missing data, then end

For each prerequisite in courseData:

If the prerequisite is not listed as a course in the file

Display the error indicating the prerequisite is missing, then end

Create a course object using courseData

Insert this course object into the hash table

Close the file and end

SearchCourse(hashTable, courseNumber):

If courseNumber exists in the hash table

Display the course's title

If the course has any prerequisites

Display "Prerequisites:"

For each prerequisite in the course's list

Display the prerequisite

If no prerequisites exist, display "No prerequisites"

Else:

Display "Course not found."

End

**Binary Search Tree Pseudocode**

LoadCourseData(fileName):

Attempt to open the file using the (fileName)

If the file cannot be opened, display the error message and end

Initialize a BinarySearchTree

While more data remains in the file:

Read data within the file

Split the data using commas and place it into the list (courseData)

If the length of courseData < 2 (missing course number or title)

Display an error message indicating the missing data, then end

For each prerequisite in courseData:

If the prerequisite is not listed as a course in the file

Display the error indicating the prerequisite is missing, then end

Create a course object using courseData

Insert the created course object into the bst

Close the file and end

searchCourse(bst, courseNumber):

Initialize currentNode to bst.root

While currentNode is not Null:

If courseNumber == current nodes courseNumber:

Display the course's title

If the course has any prerequisites:

Display "Prerequisites:"

For each prerequisite in the course's list of prerequisites:

Display the prerequisite

If no prerequisites exist, display "No prerequisites"

Else if courseNumber < current node courseNumber:

Set currentNode to leftCurrentNode

Else:

Set currentNode to rightCurrentNode

If currentNode is Null, display "Course not found."

End

**Menu Pseudocode**

Menu():

Loop:

Display "Menu:"

Display "1. Load course data"

Display "2. Print all courses"

Display "3. Print a specific course"

Display "9. Exit"

Prompt user for an option

If option == 1:

Call LoadCourseData()

Else if option == 2:

Call PrintAllCourses()

Else if option == 3:

Prompt user for course number

Call SearchCourse(courseNumber)

Else if option == 9:

Exit the program

Else:

Display "Invalid option, try again."

**Alphanumeric Order Pseudocode**

PrintAllCourses():

Sort courses by course number (alphanumerically)

For each course in the sorted list:

Display the course's number and title

If the course has any prerequisites:

Display "Prerequisites:"

For each prerequisite in the course's list:

Display the prerequisite

Else:

Display "No prerequisites"

End

**Run time 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** | 2 | 1 | 2 |
| **for each prerequisite of the course** | 1 | n | n |
| **print the prerequisite course information** | 2 | n | 2n |
| **Total Cost** | | | 5n + 2 |
| **Runtime** | | | O(n) |

**Hash Table**

|  |  |  |  |
| --- | --- | --- | --- |
| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| **for all courses** | 2 | n | 2n |
| **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** | 2 | n | 2n |
| **print the prerequisite course information** | 4 | n | 4n |
| **Total Cost** | | | 9n + 1 |
| **Runtime** | | | O(n) |

**Binary Search Tree**

|  |  |  |  |
| --- | --- | --- | --- |
| **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** | 2 | 1 | 2 |
| **for each prerequisite of the course** | 1 | n | n |
| **print the prerequisite course information** | 4 | n | 4n |
| **Total Cost** | | | 8n + 1 |
| **Runtime** | | | O(n) |

**Advantages and disadvantages**

**Vector**

Advantages:

- Extremely easy to implement

- Extremely easy to understand

- Very memory effecient when working with small data sets

- Inserting at the end can be done quickly

Disadvantages:

- Searching is slow, especially when working with large datasets

- Resizing adds can be extremely expensive

- Does not work well with large data sets that use frequent searches

**Hash Table**

Advantages:

- Extremely fast search and insertion

- Works great with large data sets that use frequent searches

Disadvantages:

- Uses much more memory than vectors due to the collision handling

- Performance can drop greatly due to the collosion handeling

- More complex to implement due to collision management

**Binary Search Tree**

Advantages:

- Ordered retrieval is extremely easy due to the data staying sorted

- Balanced trees leads to efficient search and insertion

Disadvantages:

- If unbalanced performance can drop greatly

- Much more memory is needed for the storing pointers between nodes

- Keeping trees balanced can add a sense of difficultly

**Recommendation**

After my extensive research, I strongly beleive the hash table approach is the best option for this project. It not only provides fast search and insertion but is also ideal for handling large data sets that consist of frequent searches. While it does indeed use more memory and require collision management, its speed simply cannot be over looked. The project requires course information to be retrieved and updated frequently and the hash tables fast access times will greatly improve the systems overall performance. In conclusion the hash table provides the best balance between speed, efficiency, and scalability.