**Project One**

**Joseph Wilfong**

**CS-300**

**SNHU**

**2/16/2025**

**Project One**

**Menu Options**

Start

Display Menu:

1 – Load course data from file

2 – Show all courses in order

3 – Search for a course

9 – Exit the program

Get user input

If user selects 1:

Load course data

If user selects 2:

Show all courses in alphanumeric order

If user selects 3:

Ask for a course number

Search for a course

If user selects 9:

Exit the program

Repeat the menu until the user chooses to exit

End

**Vector Implementation**

**Loading and Validating File**

Start

Open the course file

If the file cannot be opened, show an error and stop

Create an empty list for courses

For each line in the file:

Split the line into course details

If details are missing, show an error and continue to the next line

Create a course object with the course number, name, and prerequisites

Add the course to the list

Close the file

Return the list of courses

End

**Searching for Course Information**

Start

Go through each course in the list:

If the course number matches the input:

Show course details and prerequisites

For each prerequisite:

Show prerequisite information

Stop searching

If no match is found, show “Course not found”

End

**Display All Courses in Order**

Start

Sort the list of courses by course number in alphanumeric

For each course:

Show course number and name

End

**Vector Analysis**

|  |  |  |  |
| --- | --- | --- | --- |
| **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)** |

**Hash Table Implementation**

**Loading Data into a Hash Table**

Start

Open the course file

If the file cannot be opened, show an error and stop

Create an empty hash table for courses

For each line in the file:

Split the line into course details

Create a course object

Store the course in the table using its course number

Close the file

Return the hash table

End

**Finding a Course**

Start

Look up the course in the table

If found:

Show course details

For each prerequisite:

Show prerequisite information

If not found, show “Course not found”

End

**Displaying All Courses**

Start

Extract all courses from the hash table and store them in a list

Sort the list by course number in alphanumeric order

For each course in the sorted list:

Show course number and name

End

**Hash Table Analysis**

|  |  |  |  |
| --- | --- | --- | --- |
| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| Lookup course in table | **1** | **1** | **1** |
| 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** |  |  | **n + 2** |
| **Runtime** |  |  | **O(1) lookup, O(n) prerequisites** |

**Binary Search Tree Implementation**

**Inserting Courses**

Start

If the tree is empty, set the first course as the root

Otherwise, compare the new course number to the current node:

If it is smaller, go to the left

If it is bigger, go to the right

Repeat until an empty spot is found

Insert the course there

End

**Displaying All Courses in Order (Sorted Alphanumeric)**

Start

Create an empty list to store courses

Perform an in-order traversal of the BST:

Visit the left branch

Show the current course details

Visit the right branch

Sort the list by course number in alphanumeric order

For each course in the sorted list:

Show course number and name

End

**Finding a Course**

Start

Set the current node to the root

While there are still nodes to check:

If the course number matches, show details and prerequisites

If the course number is smaller, move to the left node

If the course number is bigger, move to the right node

If no match is found, show “Course not found”

End

|  |  |  |  |
| --- | --- | --- | --- |
| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| Insert course into tree | log(n) | n | O(n log n) |
| Lookup course in tree | log(n) | 1 | O(log n) |
| For each prerequisite of the course | 1 | 1 | 1 |
| For each prerequisite of the course | 1 | n | n |
| Print prerequisite course information | 1 | n | n |
| **Total Cost** |  |  | O(n log n + n) |
| **Runtime** |  |  | O(log n) lookup, O(n) prerequisites |

**How Time Complexities Were Determined**

* **Vector**
  + Searching requires scanning all courses O(n)
  + Sorting before printing takes O(n log n)
  + Sorting without takes O(n)
    - The time complexity for vector is O(n) because searching for a course requires scanning through all courses, and retrieving prerequisites also requires iteration (O(n) worst case). This makes the results total cost 4n+1, which simplifies to O(n)
  + Retrieving prerequisites involves iterating over a list O(n) worst case
  + Final Complexity: O(n log n) due to the sorting
* **Hash Table**
  + Insertions and lookups are O(1) in the average case (direct indexing via hashing)
  + Retrieving prerequisites still requires iterating over the stored prerequisites O(n) worst case
  + Final Complexity: O(1) lookup, O(n) if printing all courses
* **Binary Search Tree**
  + Searching follows the tree structure and takes O(log n) in a balanced BST
  + Inserting **n** courses requires O(n log n)
  + Retrieving prerequisites involves iterating over stored data O(n) worst case
  + Final Complexity: O(log n) lookup, O(n log n) for inserting multiple courses

**Advantages and Disadvantages of Each Data Structure**

* **Vector**
  + **Advantages:** Simple to implement, easy to iterate over all courses, dynamic resizing
  + **Disadvantages:** Searching is slow, sorting is required for ordered display, inefficient for large datasets
* **Hash Table**
  + **Advantages:** Fast lookup, efficient storage, ideal for retrieving specific course details quickly
  + **Disadvantages:** Does not maintain sorted order, performance degrades with poor has functions, requires additional memory for storing keys.
* **Binary Search Tree**
  + **Advantages:** Keeps data sorted, efficient lookup for balanced trees, allows ordered traversal
  + **Disadvantages:** More complex implementation, performance degrades to O(n) in unbalanced trees, higher memory usage

**Recommendation**

Based on the runtime analysis, the Hash Table seems like the best choice. It provides O(1) insertion and lookup, making it a good choice for retrieving course details and prerequisites quickly.