Dylan Dunagan

CS-300: DSA System Analysis and Design

15 June 2025

Module 6 Project 1

**Pseudocode**

**Vector**

Load Courses(file name)

Create vector Courses

Open CSV file

While not the end of the file

Read line

Turn line into a Course object

Add Course to vector Courses

Sort courses by their CourseId

Print All Courses(Courses)

For each course in vector Courses

Output course informantion

Print Specific Course(Courses, CourseId)

For each course in vector Courses

If input CourseId has a match

Output course information

**Hash Table**

Load Courses(file name)

Create hash table CoursesHT

Open CSV file

While not the end of the file

Read line

Turn line in a Course object

Add Course to CourseHT

Close file

Print All Hash Table Courses(CourseHT)

Sort courses in hash table by their CourseId

For each CourseId

Output course information

Print Specific Course in Hash Table(CourseHT, CourseId)

If searched CourseId exists

Output course information

**Binary Search Tree**

Load Courses(file name)

Create a search tree CourseBST

Open CSV file

While not the end of the file

Read line

Turn line into Course object

Add Course into CourseBST

Print All Tree Courses(node)

If node is not null

PrintAll(left side)

Output course information stored in each node

PrintAll(right side)

Output course information stored in each node

Print Specific Tree Course(node, CourseId)

If node is null

Return

If searched CourseId equals what is stored in node

Output course information stored in node

Else if searched CourseId is less than course stored in node

Output course information from the left side of the tree

Else

Output course information from the right side of the tree

**Main()**

Menu()

Display options

“1. Load file into data structure

2. Print All Courses

3. Print Single Course

9. Exit”

Get user input

If user input is not valid

Output error message

While user input is not “9”

Case 1

Get user input

“1. Vector

2. Hash Table

3. Binary Search Tree

9. Exit”

While user input is not “9”

Case 1:

Load data into Vector

Case 2:

Load data into Hash Table

Case 3:

Load data into Binary Search Tree

Case 9:

Exit

Case 2

Print all courses corresponding to data structure

Case 3

Prompt for Course ID

Get user input

If user input is not valid

Output error message

If user input exists

Output course from corresponding data structure

**Evaluate run time and memory of data structure**

n = number of courses

line cost = 1 unless otherwise stated

|  |  |  |  |
| --- | --- | --- | --- |
| Operation | Vector | Hash Table | Binary Search Tree |
| Insert | O(1) | O(1), O(n) | O(log n), O(n) |
| Search | O(n) | O(1), O(n) | O(log n), O(n) |
| Sort/List | O(n log n) | O(n log n) | O(n) |
| Memory Use | Compact | Higher (hashes/buckets) | Higher(pointers) |
| Ease of Use | Simple | Fast lookups | Natural order traversal |

Hash tables are the best when it comes to quick searching or looking up specific course IDs. Binary search trees provide good performance but only if the tree itself is balanced. Once the tree isn’t complete, the search performance could take significantly longer. Vectors are the easiest to use and create, but as the data set grows, the ability to search through the vector continues to increase, ultimately decreasing its efficiency.

**Advantages and disadvantages**

Vectors are the easiest to use and create. It allows for efficient iteration, allows for the vector to grow on its own dynamically as the data set grows, and allows for direct searching for elements within the vector. The downside of a vector is that as the data set grows, the ability to search for a specific element becomes more and more inefficient due to having to search each element. Also, inserting and deleting elements can be very inefficient unless you are adding or deleting it from the end.

Hash tables are the best when it comes to quick searching or looking up specific elements within the table. A user can search for a course ID, and the data structure will be able to go straight to that element. This also allows for fast insertion and deletion of elements as well. The cons of a hash table are that its performance degrades significantly when there is more than one item stored in a bucket causing collisions. Also, hash tables do not have a specific order of its elements which can also cause run time inefficiencies.

Binary search trees are very ordered and organized which allows for insertion, deletion, and searching to work well as long the tree is balanced. Once the tree is not balanced, the system will need to use more memory to be able to work its way through the tree.

**Recommendations**

I recommend the use of a binary search tree because of its efficient ordering system. In-order traversal gives a sorted list in O(n) time without previously having to sort the data structure. Also because of the binary search tree’s ordering, its search efficiency will beat a hash table or a vector especially if a larger data set is involved. Binary search trees require more memory usage than a vector, especially if unbalanced, but will always require less memory usage than a hash table. And a binary search tree also is very scalable, allowing it to grow easily as data sets grow. This works especially well if the tree is balances. Since the tree being unbalanced can pose the majority of the issues, using a self-balancing tree type can help solve those problems from the start.