A Quick Overview of Hash Tables

Let's combine all the different operations discussed previously and test out the functionality of our complete hash table class.

We'll cover the following

- Complete Implementation
- Overview

Complete Implementation#

In the previous lessons, we discussed each aspect of a hash table in detail. Below, you can find the complete hash table class.

main.py

HashTable.py

HashEntry.py

```python
from HashTable import HashTable

table = HashTable()  # Create a HashTable
print(table.is_empty())
table.insert("This", 1)
table.insert("is", 2)
table.insert("a", 3)
table.insert("Test", 4)
table.insert("Driver", 5)
print("Table Size: " + str(table.get_size()))
print("The value for 'is' key: " + str(table.search("is")))
```

```
table.delete("is")
table.delete("a")
print("Table Size: " + str(table.get_size()))

```

The `Set` class and `dict` class in Python work in the same way as the implementation above. So we have basically understood what goes on at the back-end!

Overview#

To sum up the discussion here, hash tables are the ideal data structure when you have a large amount of data and you need all basic operations to work in constant time. The challenge with hash tables comes in the form of deciding the optimized hash function. The large memory cost must also be taken care of.

Here are the time complexities for basic hash table operations:

Operation	Average	Worst
Search	`O(1)`	`O(n)`
Insertion	`O(1)`	`O(n)`
Deletion	`O(1)`	`O(n)`

In the next lesson, we will compare the performance of hash s trees.

Back
Add/Remove & Search in Hash Table (...

Next
Trees vs Hash Table

✓ Mark as Completed

Report an Issue