# Add/Remove & Search in Hash Table (Implementation)

This lesson will cover the Pythonic implementation for search, insertion, and deletion in hash tables.

In the previous lesson, we built a `HashTable` class and designed a hash function. Using that code, we will implement the main functionalities of a hash table.

## Resizing in a Hash Table #

To start things off, we will make sure that the hash table doesn't get loaded up beyond a certain threshold. Whenever it crosses the threshold, we shift the elements from the current table to a new table with double the capacity. This helps us avoid collisions.

To implement this, we will make the `resize()` function.

> Have a look at `resize()` function in `HashTable.py`.

main.py

HashTable.py

HashEntry.py

```python
from HashTable import HashTable

ht = HashTable()
# Current capacity
print(ht.slots)
ht.resize()
# New capacity
print(ht.slots)

```

Output 0.173s

```
10
20
```

## Insertion in Table #

Insertion in hash tables is a simple task and it usually takes a constant amount of time. When the hash function returns the index for our input key, we check if there is a hash entry already present at that index (if it does, a collision has occurred). If not, we simply create a new hash entry for the key/value. However, if the index is not `None`, we will traverse through the bucket to check if an object with our key exists.

It is possible that the key we are inserting already exists. In this case, we will simply update the value. Otherwise, we add the new entry at the end of the bucket. The average cost of insertion is O(1). However, the worst case is O(n) as for some cases, the entire bucket needs to be traversed where all n elements are in a single bucket.

After each insertion, we will also check if the hash table needs resizing. The `threshold` will be a data member of the `HashTable` class with a fixed value of $0.6$.

> Have a look at `insert()` function in `HashTable.py`.

main.py

HashTable.py

HashEntry.py

```python
from HashTable import HashTable

ht = HashTable()
ht.insert(2, "London")
# Collides with (2, "London") - Added next to it at the same i
ht.insert(12, "Moscow")
ht.insert(7, "Paris")

print("Size of the list: " + str(ht.size))

```

Output 0.233s

```
2 - London inserted at index: 2
12 - Moscow inserted at index: 2
7 - Paris inserted at index: 7
Size of the list: 3
```

## Search in a Hash Table #

One of the features that make hash tables efficient is that search takes *O(1)* amount of time. The `search` function takes in a `key` and sends it through the hash function to get the corresponding index in the table. If a hash entry with the desired key/value pair is found at that index, its value is returned. Search can take up to *O(n)* time, where **n** is the number of hash entries in the table. This is possible if all values get stored in the same bucket, then we would have to traverse the whole bucket to reach the entry.

Have a look at `search()` function in `HashTable.py`.

main.py

HashTable.py

HashEntry.py

```
from HashTable import HashTable

ht = HashTable()
ht.insert(2, "London")
print(ht.search(2))
print(ht.search(7))

```

```
Output                                                    0.174s

 2 - London inserted at index: 2
 London
 None
```

## Deletion in Table #

Deletion can take up to *O(n)* time where **n** is the number of hash entries in the table. If they all get stored in the same bucket, we would have to traverse the whole bucket to reach the entry we want to delete. The average case, however, is still O(1).

Have a look at `delete()` function in `HashTable.py`.

main.py

HashTable.py

HashEntry.py

```
from HashTable import HashTable

ht = HashTable()
ht.insert(2, "London")
ht.insert(7, "Paris")
ht.insert(8, "Cairo")
print("size:", ht.get_size())
ht.delete(2)
print("size:", ht.get_size())
ht.search(2)

```

```
Output                                                    0.441s

 2 - London inserted at index: 2
 7 - Paris inserted at index: 7
 8 - Cairo inserted at index: 8
 size: 3
 2 - London deleted
 size: 2
```

In the next lesson, we will bring all these functions together to create our complete `HashTable` class.

Back

Building a Hash Table from Scratch

Next

A Quick Overview of Hash Tables

✓ Mark as Completed

Report an Issue

Ask a Question (https://discuss.educative.io/tag/addremove--search-in-hash-table-implementation__introduction-to-hashing__data-structures-for-coding-interviews-in-python)