Building a Hash Table from Scratch

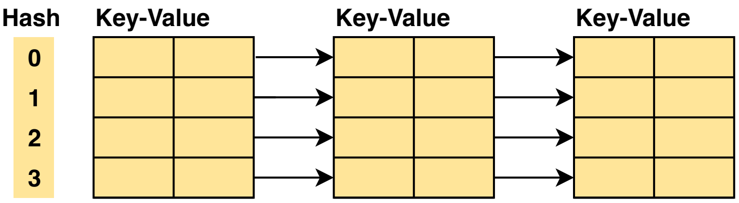
Learn about how hash tables are implemented in Python.

We'll cover the following

Hash Table Using Bucket Chaining

As said earlier, hash tables are implemented using lists in Python. The implementation itself is quite simple. We will use the chaining strategy along with the resize operation to avoid collisions in the table.

All the elements with the same hash key will be stored in a linked list at that index. In data structures, these lists are called **buckets**. The size of the hash table is set as $n*m$ where n is the number of keys it can hold and m is the number of slots each bucket contains. Each slot holds a key/value pair.



Implementation

We will start by building a simple `HashEntry` class. As discus[illegible]ar[illegible]a typical hash entry consists of three data members: the **key**, the **value**, and the **reference to a new entry**. Here's how we will code this in Python:

HashEntry.py

```python
class HashEntry:
    def __init__(self, key, data):
        # key of the entry
        self.key = key
        # data to be stored
        self.value = data
        # reference to new entry
        self.nxt = None

    def __str__(self):
        return str(entry.key) + ", " + entry.value

entry = HashEntry(3, "Educative")
print(entry)

```

Now, we'll create the `HashTable` class which is a collection of `HashEntry` objects. We will also keep track of the total number of **slots** in the hash table and the **current size** of the hash table. These two variables will come in handy when we need to resize the table.

Here is the basic implementation in Python:

HashTable.py

```python
class HashTable:
    # Constructor
    def __init__(self):
        # Size of the HashTable
        self.slots = 10
        # Current entries in the table
        # Used while resizing the table when half of the table gets filled
        self.size = 0
        # List of HashEntry objects (by default all None)
        self.bucket = [None] * self.slots
    # Helper Functions

    def get_size(self):
        return self.size

    def is_empty(self):
        return self.get_size() == 0


ht = HashTable()
print(ht.is_empty())
```

The last thing we need is a hash function where a hash function maps values to a slot in the hash table. We tried out some different approaches in the previous lessons. For our implementation, we will simply take the modular of the key with the total size of the hash table (slots).

```
# Hash Function
def get_index(self, key):
    # hash is a built in function in Python
    hash_code = hash(key)
    index = hash_code % self.slots
    return index
```

Our hash table is now ready. As always, the next step is to implement the operations of *search*, *insertion*, and *deletion* one by one. We will cover this in the next lesson. Stay tuned!

companies apply to you instead of you applying to them. how ⓘ

← Back

Collisions in Hash Tables

Next →

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

☑ Mark as Completed

Report an Issue