Hash Table (Implementation)

create a hash table, add a hash, search for a key (Reading time: 2 minutes)

To implement a hash table, we create a class.

```
class HashTable {
  constructor() {
    this.values = {};
    this.length =  0;
    this.size =  0;
  }
}
```

The constructor contains an object in which we're going to store the values, the length of the values, and the entire size of the hash table: meaning how many buckets the hash table contains. Somewhere in these buckets, we can store our data.

Next, before we can do anything else, we need to implement our hashing function. One example of a hashing function:

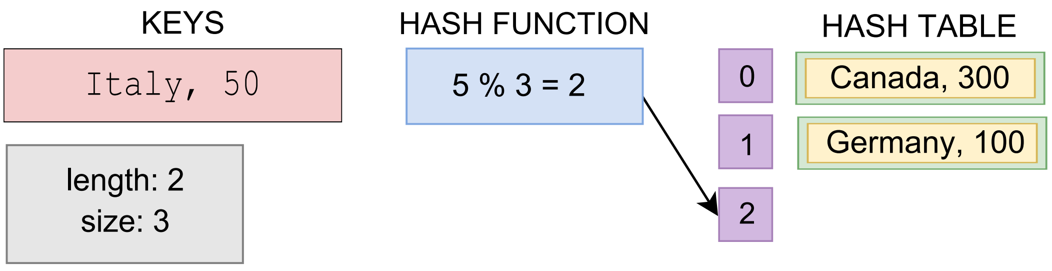
```
calculateHash(key) {
  return key.toString().length % this.size;
}
```

Let's say we have the string "Hello" and we create a hash table with the size of 10. "Hello" has the size of 5, so 5 % 10 becomes **5**! The string "Hello" is now stored in the bucket with hash 5. If later on, we want to see where the key/value pair with the key "Hello" has been stored, it would return the same hash as the length of "Hello" hasn't changed, and neither has the size of the hash table!

```
add(key, value) {
   const hash = this.calculateHash(key);
   if (!this.values.hasOwnProperty(hash)) {
      this.values[hash] = {};
   }
```

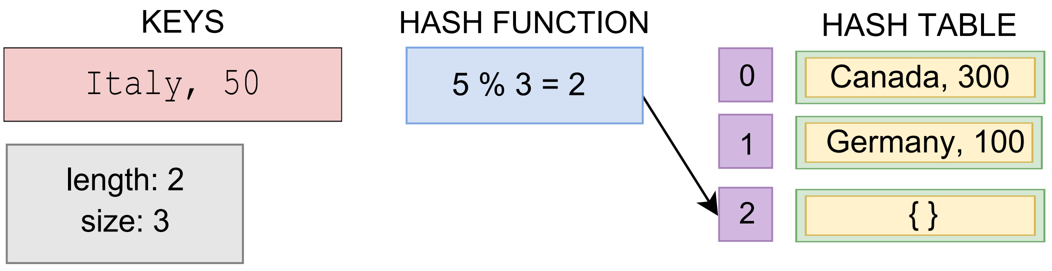
```
    if (!this.values[hash].hasOwnProperty(key)) {
      this.length++;
    }
    this.values[hash][key] = value;
  }
```

To add a key/value pair, we first need to calculate the hash with the key provided. If this hash is brand new (meaning that no other key/value pair used it yet and it's not in the values object), we initialize an empty object for that hash. Next, we check whether the hash has a property with the same key name! If that's not the case, it means we will add a new key/value pair, and the length of the hash table grows. Lastly, we add the key/value pair to the right hash.



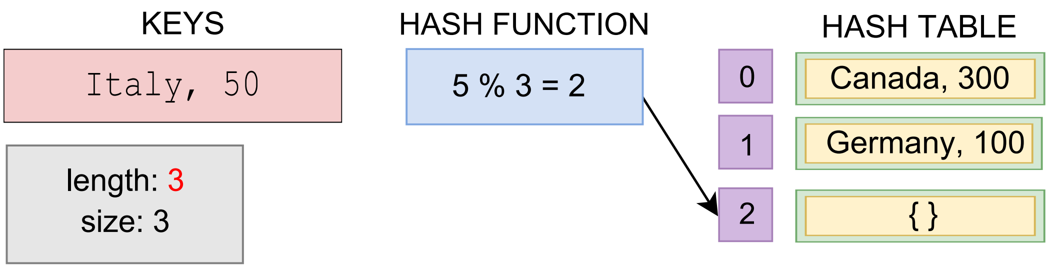
The length of "Italy" is 5, so the hash function returns the hash 2.

1 of 4



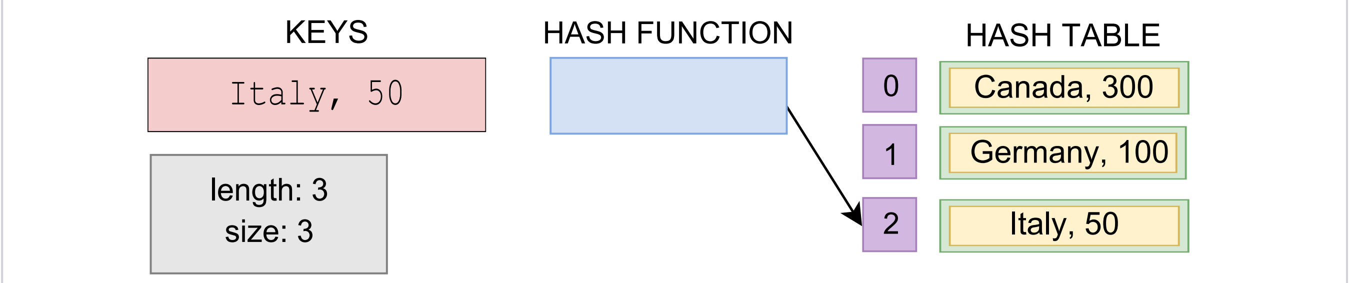
!this.values.hasOwnProperty(hash) returns true, so we initialize an empty object.

2 of 4



The hash's key value on the object doesn't have the value yet, so we increment the length by one.

3 of 4



We set the key equal to the value in the hash table, which adds it.

4 of 4

Searching in a hash table goes very fast. As with an array we have to go through all of the elements until we find it, with a hash table we simply get the index. This means that its runtime is constant, O(1).

```
search(key) {
  const hash = this.calculateHash(key);
  if (this.values.hasOwnProperty(hash) && this.values[hash].hasOwnProperty(key)) {
    return this.values[hash][key];
  } else {
    return null;
  }
}
}

//create object of type hash table
const ht = new HashTable();
//add data to the hash table ht
ht.add("Canada", "300");
ht.add("Germany", "100");
ht.add("Italy", "50");

//search
console.log(ht.search("Italy"));
```

Searching for the key "Italy" in the above hash table

First, we calculate the hash again. As the length of the string and the size of the hash table haven't changed, the hash remains the same. Then, we check whether the hash is within the object, and whether that hash points to the key

we’re looking for. If that’s the case, return the value that that pair stores, else nothing gets returned.

In the next lesson, I will talk about the time complexity of the various functions of the hash table.