Notes - Algorithms & Data Structures - Hash Tables - Part 3

- Dr Nick Hayward

A brief intro to *hash* tables and hash functions.

Contents

Intro

Hash tables are a particularly useful, and fast, data structure.

From a conceptual perspective, we may define a hash table data structure as follows

- store each item in an easily determined location
    - so no need to search for item
- no ordering to maintain
    - for insertion and deletion of items

As such, this data structure has impressive performance, as far as time is concerned. However, there is a tradeoff with additional memory requirements, and conceptually harder implementation for custom patterns.

Hash function

Load factor is, of course, an important consideration for usage and management of a hash table. However, this is not possible without a good hash function.

In effect, a good hash function should try to evenly distribute values in the underlying array.

By comparison, a poor hash function will create groups of values, thereby producing many collisions in the hash table.

We may never need to write a hash function from scratch, but a good example is the **SHA function**.

SHA function

As we use a hash table, we need a good hash function to determine where to assign a data element in an array. In effect, it should work out even distribution to optimise load factor, and try to avoid collisions as much as possible.

If this works correctly, we should be able to perform constant-time lookups for the hash table. This good hash function allows us to quickly check the value of a key. In effect, it returns the index of the array to check in `O(1)`, constant time.

So, a *secure hash algorithm* (SHA) function is an example of a good hash function we might choose to adapt for our hash table.

For example, if we pass a string such as `hello` to *SHA* it will return a hash

```
'hello' -> 4dg54ab...
```

So, SHA is a hash function, which generates a hash, or a short string. As expected, SHA will generate a different hash for every string.

Common usage is to check and validate files, for example in file sharing, project usage &c. This can be particularly useful for very large files.

For example, two users may need to check and verify they're using the same file, even though they may have separate copies. SHA is used to calculate the hash, and each user may then check their file against the hash.

Likewise, SHA is also useful for verification of passwords, and associated cursory encryption. SHA may be used to compare strings without revealing the original string content. In a database, we may store the generated SHA hash instead of the original password string.

To check and use these passwords, we may hash the input string and then check it against the saved hash in the database. In effect, we're only comparing the hashes, and not the original string passwords.

Another benefit of this use of SHA is that the hash is *one way*. We may get the hash, but not the original string from the hash.

locality insensitve

Another useful and important feature of SHA usage is its lack of locality sensitive hashing.

For example, if we consider the following string

```
daisy -> hu9m362g...
```

If we then modify the string by even a single character, and then generate the hash, SHA will return a new, different hash.

```
daily -> h4dg96hj...
```

Of course, a clear benefit of this approach is that we can't now compare hashes to check for reverse engineering the hash. In effect, hashes can't be compared to iteratively return the original string.

locality sensitive

However, there may be instances where we actually need such *locality sensitive* hash functionality. This is where we may consider **Simhash**.

If we modify a string and then generate a hash using *Simhash*, it will generate a hash that is only a slight update to the previous hash. The benefit is that we may use this to compare hashes, and thereby determine the proximity of two strings. For certain use cases, this can be particularly useful.

For example, collation of texts, web crawlers &c. may use this approach. If we check various online sources, we may then use *Simhash* to identify duplicates.

Likewise, editors, teachers, and anyone who wants to check various textual sources may use *Simhash* for this collation.

Verification of copyrighted material is another sample use for *Simhash*.

SHA family

SHA is a group of algorithms we may use for hashing values. For example,

- SHA-0
- SHA-1
- SHA-2
- SHA-3

So, if we need to use SHA to hash passwords, and other sensitive data, we may commonly use SHA-2 or SHA-3.

Further details are available at the following URL

- SHA algorithms - Wikipedia