Hashtables

Sergios Anestis Kefalidis
Konstantinos Nikoletos
Kostas Plas
& Manolis Koubarakis

What’s a hashtable?

A hash table, also known as a hash map, is a data structure that allows efficient insertion, deletion, and retrieval of key-value pairs. It uses a hash function to map each key to a unique index in an array, where the corresponding value is stored.

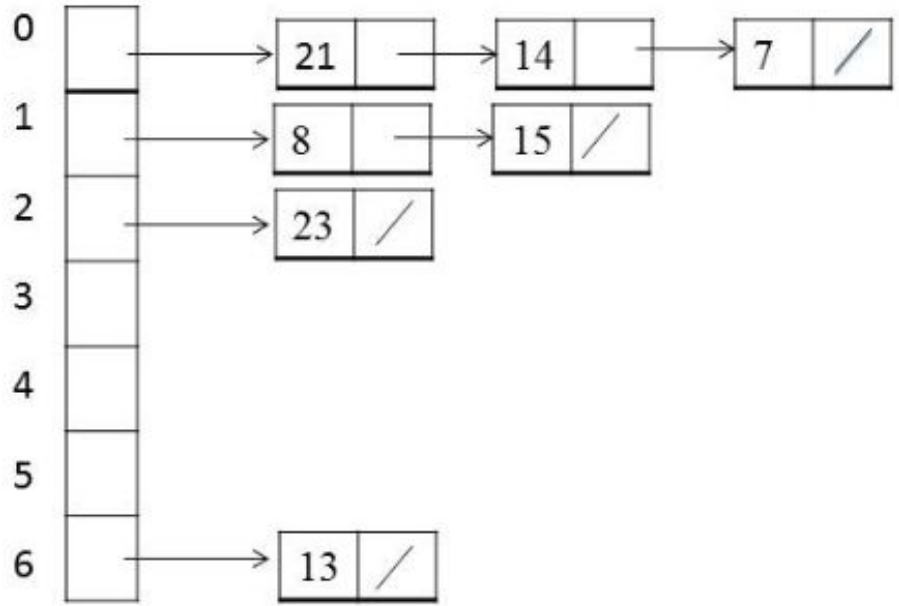
Properties:

1. O(1) complexity Search/Insertion/Delete in the best case
2. Uses a hash function to map each **key to a unique index** in an array, where the corresponding value is stored

Hashing with chaining

Consists of:

- Table of size N
- Each table cell is a List



Hash function

First we need to define a method that creates a hash (key) from a given string.

One simple example:

```
hashFunction(string):
    hash = 0
    for char < string_length:
        hash += string[char]   // add ascii-code value
    return hash % hashtable_size
```

Insertion in a Hashtable

For example: Lets insert Customer with id “A1” and <”John”, “Doe”, etc> in a hashtable with size = 5

1. We take its unique key and calculate the corresponding hash:

   hashFunction(“A1”) = 4

2. Find the cell table corresponding in this hash: table[hash]
3. Add the customer to the list in table[hash] position

There’s a problem. Can you guess what it is?

Load factor

Problem: if the size of the hashtable is significantly smaller than the number of elements inserted then the time complexity rises as we search in lists. SO:

- In each insertion we calculate the load factor:

  Load factor = Number of elements in HT / Size of hashtable

- If the load factor is > than 0.75 or 0.8 we do resing!

For example:

- 4 customers in a hashtable of size 5

  Load factor = ⅘ = 0.8

Resize

Create a new hashtable with size:

new_size = first_prime(2*hashtable_size)

For example: A hashtable with size = **10** will transform to hashtable with size equal to **23** [new = first_prime(20) = 23]

Method:

1. We traverse all lists in the old hashtable and re-insert to the new hashtable.
2. Destroy the old hashtable and return pointer to the new one.

Hashtable lookup

Let’s say we are looking record with id “A5”

Method:

1. We hash “A5” key
2. Take the list in the hashFunction(“A5”) position of the array
3. Search in list for this key
4. Return entity with key “A5” or NULL if it doesn’t exist

Hashtable remove entity

Let’s say we want to remove record with id “A5”

Method:

1. We hash “A5” key
2. Take the list in the hashFunction(“A5”) position of the array
3. Search in list for this key
4. Remove entity with key “A5” or NULL if it doesn’t exist