Hashing

Created	@September 12, 2022 8:15 PM
Class	
Type	
Materials	
Reviewed	☐

In computer science hashing is a very important concept. There are n number of problems and applications that depend on hashing. In order to understand hashing, we need to first get clarity on the term `encryption`

Encryption

It is the process of encoding information. We can transform a piece of data into another form which is not human understandable. So encryption generally goes 2 ways , i.e. you can encrypt the data and then decrypt it as well.

Hashing

Before we understand about what is hashing, we need to understand that generally hashing is one way, i.e. we can hash the data, but cannot retrieve original data back from the hash.

What is hashing ?

Hashing means mapping a large set of data, into a smaller set of data.

When I say mapping a large set of data to a small set then I mean something like this.

Now how do we map ? is there some mechanism ?
So we define a term hash function, which is used to map this data.

Hash function

Let's say we have a function `y = f(x)`

Then `f` is called the hash function which takes `x` as the input which is a data value from the larger set and returns a value `y` which belongs to smaller set.

There can be different implementations of hash function, let's take one as an example:

`f(x) = x%100`

So for example: x = 88 → f(188) = 188%100 = 88

Now there is an interesting part, here,

f(188) = f(288) = f(388) …….. = 88 → collision

What is collision ?

In hashing if two or more values are mapped to a same value due to hash function, we call this phenomenon as Collision.

Application of Hashing:

In programming languages we have things like Hashtable (java) , dictionary (python), object, map (JS) etc, are based on the concept of hashing

How to avoid collision ?

If we try to make a strong hash function then we can reduce the possibility of collision. And there are few mechanisms to avoid collision to some extent.

1. Open addressing → In open addressing whenever collision happens we try to find a new mapping for the value.
    a. Linear probing → it does a sequential search in case of collision to find an available mapping. if we got a mapped value i, and there is already a value mapped to i, then we search for availability of i + 1 then i + 2 then i + 3 and so on
    b. Quadratic probing → it does a quadratic jump based search in case of collision to find an available mapping. if we got a mapped value i, and there is already a value mapped to i, then we search for availability of i + 1^2 then i + 2^2 then i + 3^2 and so on
    c. Double hashing - we keep 2 hash functions, h1 and h2, we first try to use h1 and get a mapped value say i, if i is already mapped to some other value we search for i + h2(x) then i + 2*h2(x) then i + 3*h2(x) and so on
2. Separate Chaining

Example hash function:

1. f(x) = x%m
2. Polynomial hash function → for some value of x and m

    f(1257) = (1x^3 + 2x^2 + 5x + 7)%m
3. djb2 hashing

http://www.cse.yorku.ca/~oz/hash.html

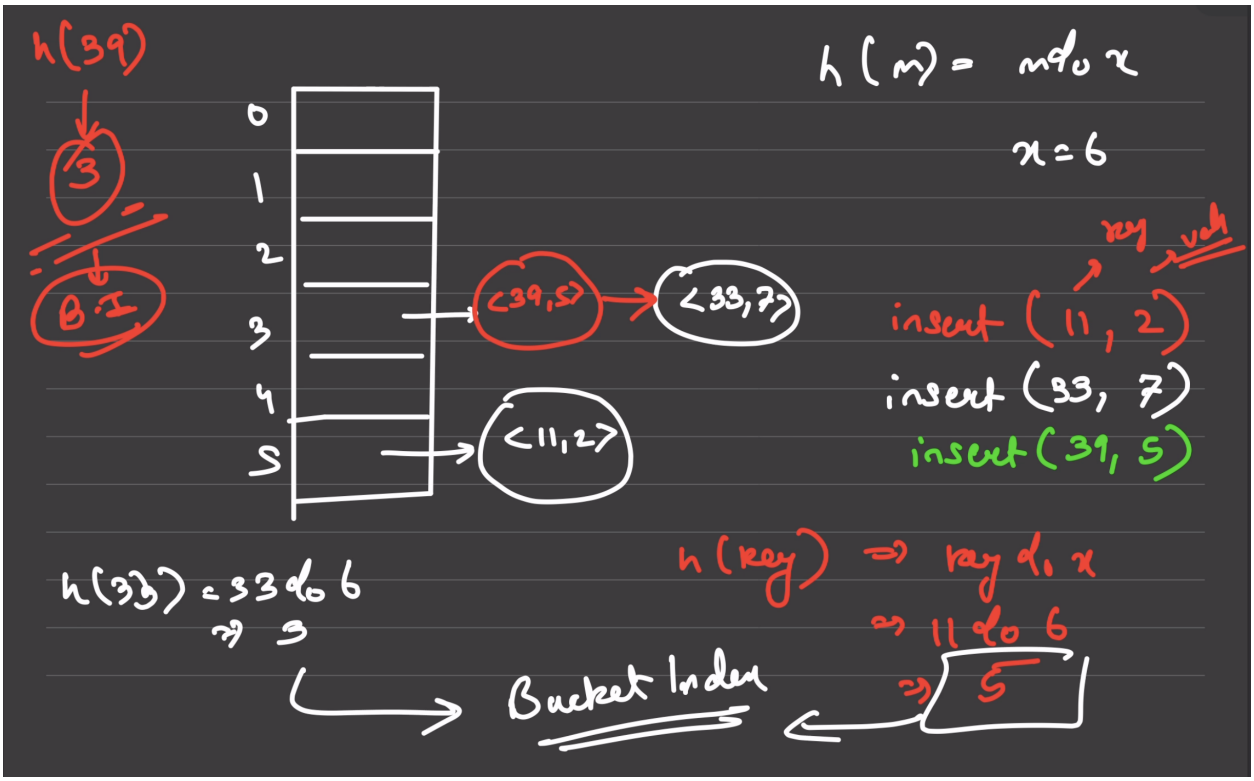
How can we implement Hashmap / dict / js object, map …

Internally hashmap is based on the concept of hashing and it is implemented in the form of array of linked list.

There is a primary array of length `x` and every index might contain a linked list.

The actual values stored inside Hashmap are <key, value> pairs.

To store these <key, value> pairs we pass the key through the hash function, which returns us a bucket index. This bucket index is the index of the array on which we will add this <key, value> pair as the node of a linked list.



Now we know how to store, but how can we retrieve the value for the key ?

We can again pass the key into the hash function, get the bucket index, and then do a linear search on the linked list.

You might be thinking, **Linear Search !!!!! Really !!!!!**

The fact is that we will keep linked lists of extremely small size. This will make linear search almost constant.

How will we do it ?

So two things are going to help us:

- Lambda factor
- Rehashing

Lambda / Load factor:

It is the ratio of the total number of nodes present in the array of linked list divided by the size of the array,

What is the significance of this ratio ?

Total number of nodes in the array of linked list represents total number of elements (<key, value>) inserted in the hahsmap.

Size of the array represents, the max number of entries we could have taken without collision.

So the ratio of the total number of elements to the size of the array, denotes how much the hashmap is filled ?

So lambda factor represents how much the array is filled w.r.t it's capacity.

λ = (number of elements inserted) / (size of hashmap)

Generally we keep 1.0 ≥ **λ ≥ 0.5**

Rehashing

Note: In hashmap when we decide a hash function, whatever hash function we take, we do a modulo by the size of the array.
What is rehashing ? In rehashing, whenever we hit a condition where the ratio of the number of elements inserted to the size of the array, exceed **λ factor,**

We create a new array of size double the previous array, then insert all the elements of the previous array of LL, to this new double sized array of linked list, but to insert the prev values, we again make them go through the hash function. And because the size in which they will be inserted is double the hash function's modulo will also be doubled.