Hashing

- Process of converting string / key into another value

string / key → Hash function → Hashed value

Usage:

1. used in hash tables for data storage & fast lookups ← Hash Maps, almost O(1)
2. used in zipping / condensing files
3. used in cryptology / encryption

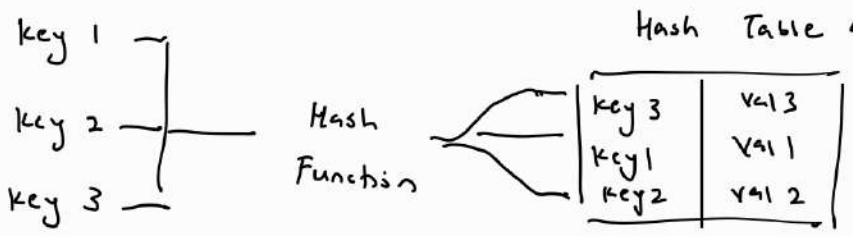
} can vary complexity based on use case

Hash function / algorithm

- convert a → b
- depending on use case, may be easy from a → b but difficult from b → a

1. Hashing for Data Storage (Hash Table)

- Hashing algorithm not so complex (Hash to get index)
- Hash table = data structure with faster search time

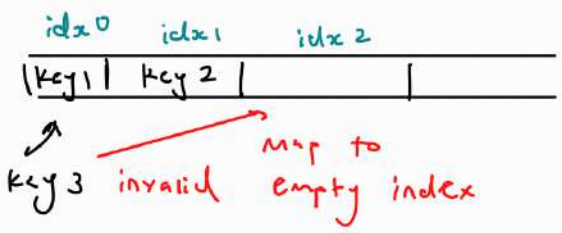


Hash Table ← can be implement with arrays

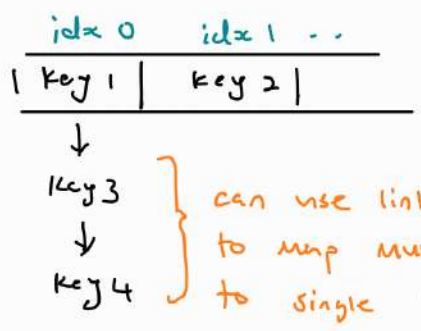
↳ keys will be mathematical value
vals will be data stored in table

Collisions

- Open addressing: Single key map to single array index
  ↳ if existing index has key, the new key must find next avail index.



- Closed addressing: Multiple keys may map to single array index using linked list



} can use linked list to map multiple keys to single index

- Load Balance (Hash) : scaling up hash table
  ↳ before hash table gets too full
  - collisions decreases efficiency.

Hash Table complexity

Searching, Insertion, deletion: Best & Avg case = O(1), worst case: O(n)

↑ during load balance (scale up) need to rehash every element or consistent collision due to fullness)

Linear Probing

- if index already taken, move to next index., repeat until empty found.

disadvantage: if too much linear probing, becomes O(n)

└ prevention:

- small load (Hash table empty)
- fully random set of keys to reduce likelihood of collision

Quadratic Probing

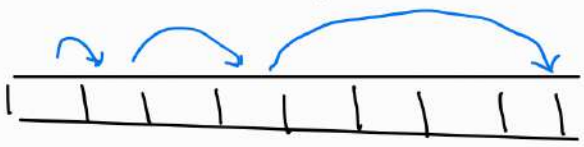
- reduce likelihood of clustering ← Bunching of keys together
- Don't perform adjacent probing., but by quadratic differential.
- Have a separate Algo for probing

└ eg: check 1 space, then 2, then 4 then 8 etc.

eg: Hash(25) % 10 = 3 ← idx

if idx taken, try:

(3+1*1) % 10 = 3
(3+2*2) % 10 = 7
(3+3*3) % 10 = 2
⋮



} Jumping of probing prevents clustering. (or slows it down)

Rehashing

- Another solu to collisions.
- Hashing more than once to introduce randomness

└ using same original hash algo multiple times. eg: (x * 3) % 18

└ use a separate collision hashing algo. eg: (x*2 +3) % 18

* Need to beware of loops: eg: (x * 2) % 10.

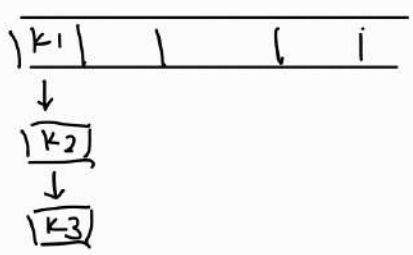
if 5: → 5 * 2 % 10 = 0
0 * 2 % 10 = 0
⋮

or algo that repeats certain number only

usually happens in simple algos

Closed Addressing

- Allow multiple key in single index (no collisions)
- more space consumed.
- Linked list used to store all the keys held by index.



issue: if linked list too large, same issue as collision. Need to probe every key.

Hash Table eg.

- Database primary key as unique key. → Hash algo → stored in table.

eg: djb2 Hash fn.

- Near O(1) for search / accessing

Hashing vs Encryption

Encryption

- 2 way possible (hash / unhash)
- reversible process
- want to transfer data with other parties
- uses encryption algo
  (AES encryption algo)

Hashing

- 1 way only
- irreversible process (By definition)
- More secure than encryption

use case

- checksum
- password storage
- idxing (storage)

* still susceptible to de-hashing
  - eg: using hash table (then lookup)
    hash to reverse
    ↓
    countered with Salting

- uses hash fn