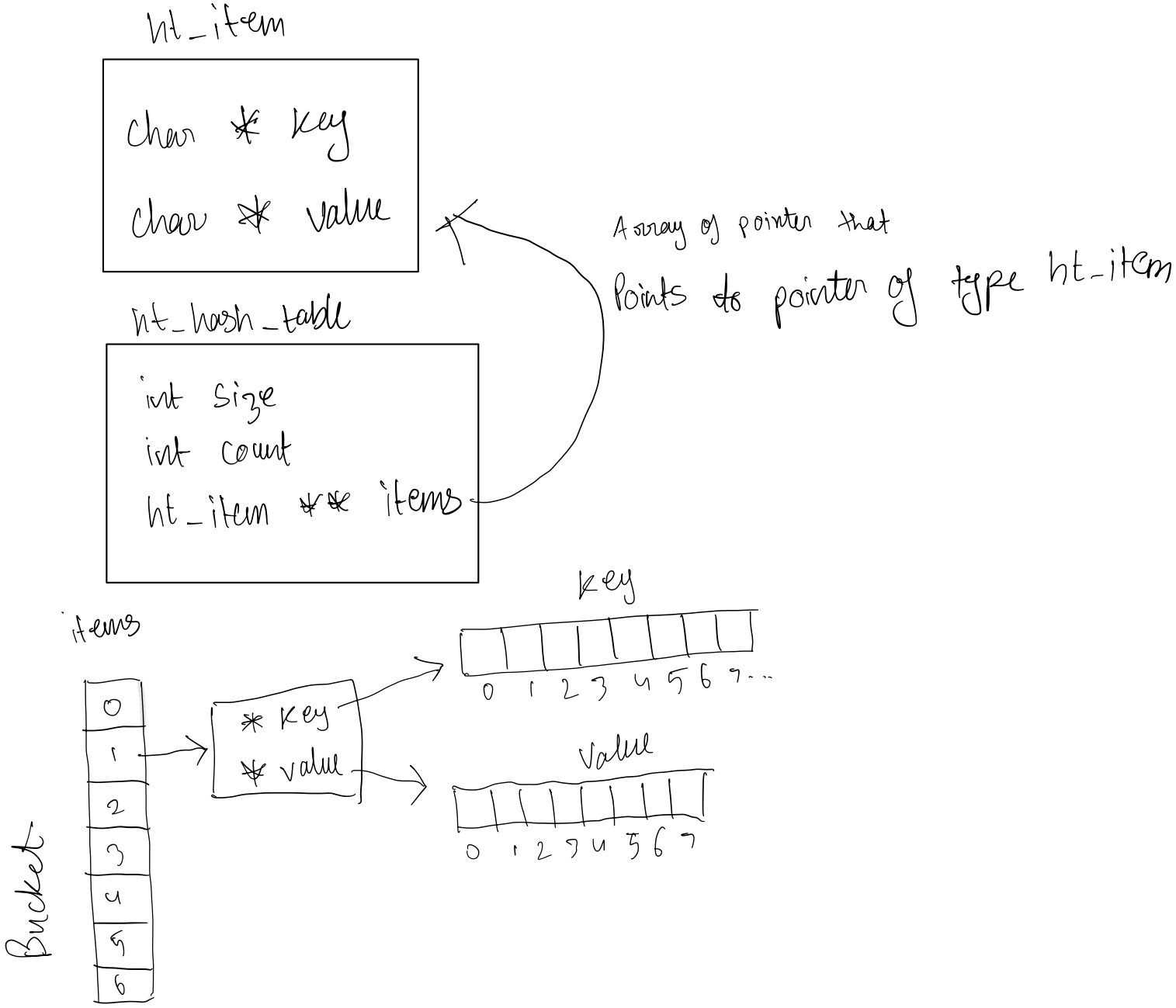
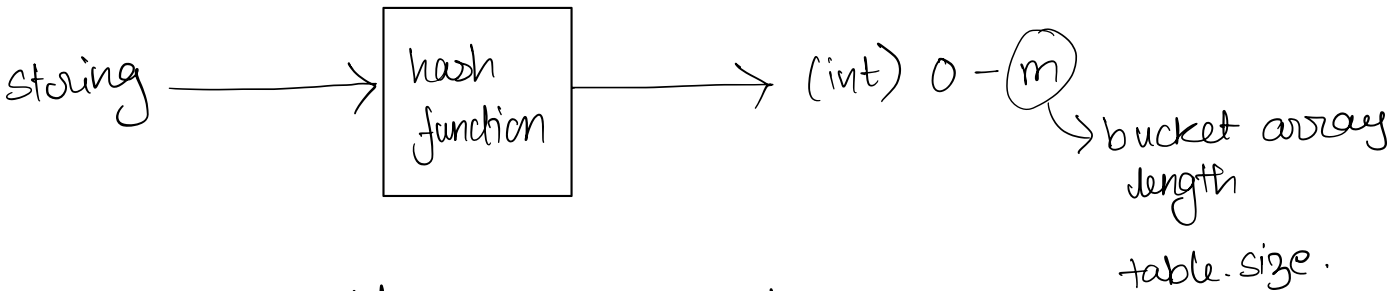


what does a hashing function do?



The hash function should return an evenly distributed indexes for an average set of inputs

what does even distribution mean?

It means that the hashing function should be created in such a manner that same output for different input can be minimised.

why minimise?

If we get same output from the hash function for different sets of input, collisions can occur.

Resolving these collisions takes time and thus reduces the efficiency of our hash table.

hash function

→ convert the storing to a large integer
→ Reduce the size of that integer to a fixed range by taking its remainder (% table→size)

generic storing hashing function

hash += a^(length_storing − (i+1)) * S[i]

a → Prime number greater than the alphabet (>128)

Pathological Data

Pathological set of inputs that all hash to same value.

How to find these inputs?
Run a large set of inputs through the function. All inputs that hash to a particular value form a pathological set.

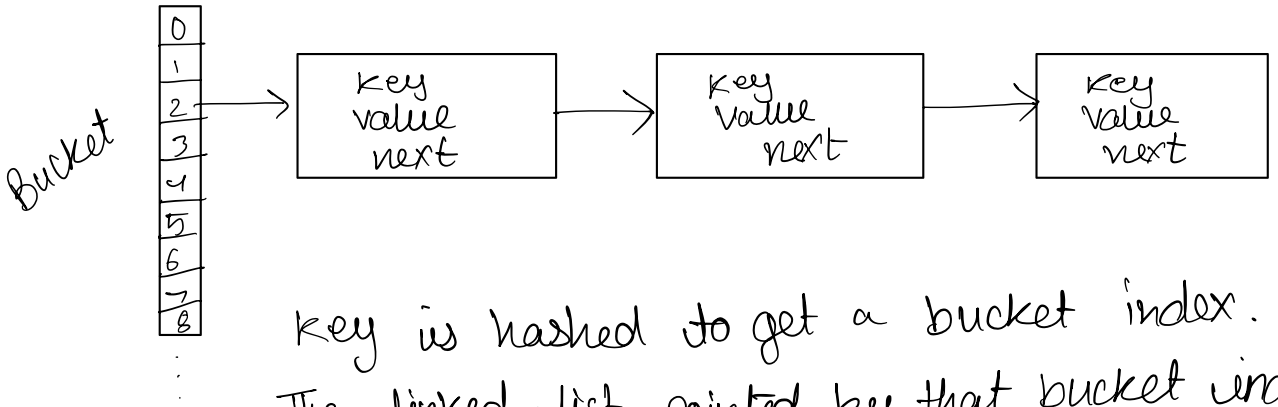
Why do these inputs matter?
Searches for these keys take longer (O(n)) than normal (O(1)).
Large set reduces the efficiency of hash table.

Malicious users can use these keys as a denial of service attack against the system.

what happens when collisions happen?
how to resolve collisions?

(1) seperate chaining

each bucket contains a linked list.



key is hashed to get a bucket index.
The linked list pointed by that bucket index is traversed to find the key.
Traversing the key takes O(n) time.

(2) open Addressing.

when collisions happen, the collided item is placed in some other bucket.
solves the space inefficiency of seperate chaining.

(2.1) Linear probing.

when collision occurs, the index is incremented and the item is put in next available bucket.

when searching, the key is checked against the bucket index given by the hashing function. If key matches, the value is returned else the index is incremented till the key is found.

Takes time to search the key position.

2.2 Quadratic Probing.

Similar to linear probing but instead of putting the collided item in the next available bucket, we try to put it in the bucket whose indexes follow the sequence:

$$i, i+1, i+4, i+9, i+16 \ldots\ldots$$

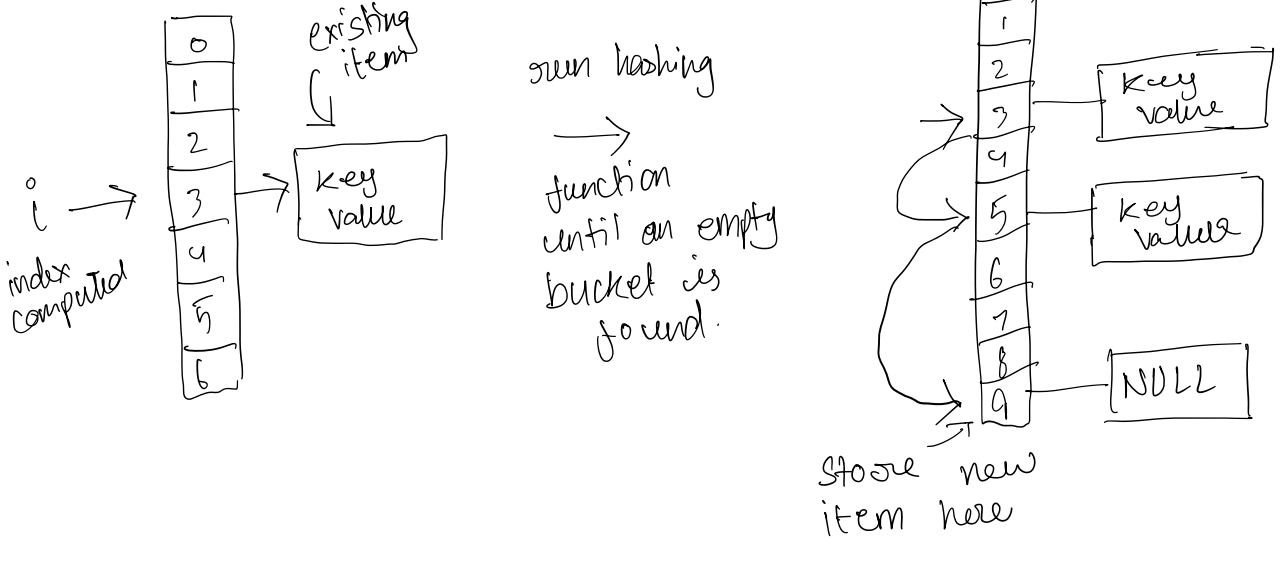
This reduces but does not remove clustering.

2.3 Double hashing

Aims to solve the clustering problem.
We use second hash function to chose new index.

Insert operation

Allocates memory for the new item to be stored.
Uses hashing function to compute an index to store the item pointer at.
If the index computed already has an item run a while loop that keeps computing the index until an empty bucket is found.



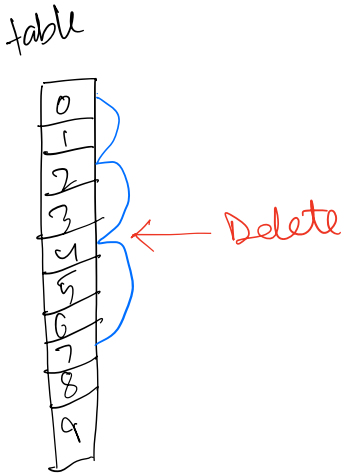
Search operation

Similar to insertion, but we check the key at each iteration of while loop to check whether the item's key matches the key we are searching for.

If no key is found return NULL.

Delete operation

The item we wish to delete may be part of collision chain



if we delete the item at index 4, next time a search operation is performed for key placed at bucket index 7 will be marked as not found.
As the hashing function reaches the bucket index 4 it will see that the bucket is empty and mark it as not found.

To avoid this we replace the item we wish to delete with a global sentinal item that represents deleted item in the list.

Updates in other functions:

1. search.
   we will compare the key only if it is not a global sentinal item.
2. insert
   we will insert even if the item is a global sentinal item.

Resizing

Currently the hash table has a fixed number of buckets. As more items are inserted, the table starts to fill up and increases the rate of collision.

We need to resize the table to reduce collisions and increasing the overall capacity of the table.

On every insert or delete operation we calculate the tables load. (ratio of filled buckets to total buckets)

if load > 0.7 → upsize the table

if load < 0.1 → down size the table.

We create a new hash table roughly twice or half the size of the existing table. and insert all non deleted items into it.

New Array size should be a prime number. double or half the current size.

suppose base size is 50

we find the first prime number larger than 50 and use it as actual size of the array.

For up size → double the base size and find next largest Prime number

For down size → half the base size and find next largest Prime number.