**Objective-** To understand the concept of

* Hashing
* Methods of hashing
* Collision
* Collision resolution techniques

**Theory-**

**Hash Tables-**

* It is like an array, with a collection of data items, where the data is being stored in the particular position not like the one seen in normal arrays.
* Unlike an array hash tables need to have hash function to compute the position of the element to be stored.
* So hash structure allows to retrieve the information directly.

**Hashing-**

* Key to address transformation
* Hashing is one of the searching scheme.
* Suppose there is a company of 100 people but EMPID are 00000 to 99999
* So here it is not possible to take array size 100,000
* So we will use only last two digits as EMPID
* Here elements are not stored according to value of key.
* But elements are stored with respect to some function of the key value i.e. called Hash value
* And searching technique we are using is called Hashing Technique.
* Efficiency of hash table lies with selection of hash function**.**

**Hash Function-**

* The position of the key in the array is given by a function called Hash Function.
* If suppose size of key is very large Ex. 32 bit integer then |k|=2 32. But actual number of items stored in the container are significantly less than |k| then Hash function is used.
* So here we need function H:k->{0,1,…..M-1}
* This function maps the set of values to be stored in the container to subscript in the array of length M.
* This is called Hash Function.
* There may be many keys x and y such that x!=y but h(x)=h(y). This situation is called a collision

**Different hashing techniques**

1. **Division-** by prime no.
2. **Folding-** portions of the key are often recombined, or Folded together
3. **Mid-square function** square the key, then use the middle part as the result
4. **Radix transformation**- change the base-of-representation of the numeric key, mod by table size
5. **Extraction-** use only part of the key to compute the result

**Collision & Resolving Collision:**

* Here hash(key)=key % 100
* For key values 33542 & 13942 location is same (42)!!!
* It is called as collision.
* **For avoiding collision different methods are used**.
* Open Addressing

1. Linear probe
2. Quadratic probe
3. Key offset (Double Hashing)
4. Pseudorandom

* Linked List/ Separate Chaining
* Bucket hashing

**Direct hashing-**

* Key itself is same as memory location
* Range of keys must be known in order to allocate memory
* Not possible for non-integer keys
* No possibility of collision

**There are two broad ways of collision resolution:**

**1. Separate Chaining:: An array of linked list implementation.**

**2. Open Addressing: Array-based implementation.**

(i) Linear probing (linear search)

(ii) Quadratic probing (nonlinear search)

(iii) Double hashing (uses two hash functions)

**Separate Chaining**

The hash table is implemented as an array of linked lists.Inserting an item, r, that hashes at index i is simply insertion into the linked list at position i. Synonyms are chained in the same linked list

**Open Adressing-**

All items are stored in the hash table itself. In addition to the cell data (if any), each cell keeps one of the three states: EMPTY, OCCUPIED, DELETED.While inserting, if a collision occurs, alternative cells are tried until an empty cell is found.

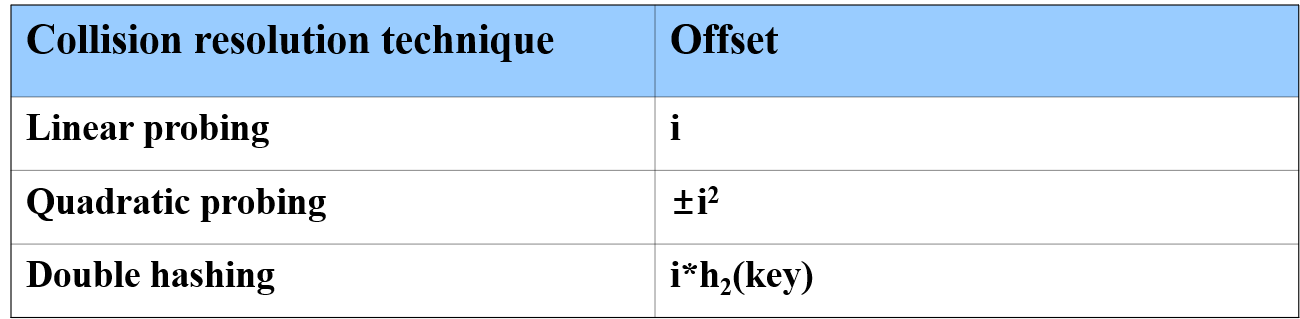
Deletion: (lazy deletion): When a key is deleted the slot is marked as DELETED rather than EMPTY otherwise subsequent searches that hash at the deleted cell will fail.

Probe sequence: A probe sequence is the sequence of array indexes that is followed in searching for an empty cell during an insertion, or in searching for a key during find or delete operations.

The most common probe sequences are of the form:

**hi(key) = [h(key) + offset] % n,**

where h is a hash function and n is the size of the hash table



**Linear Probing**:

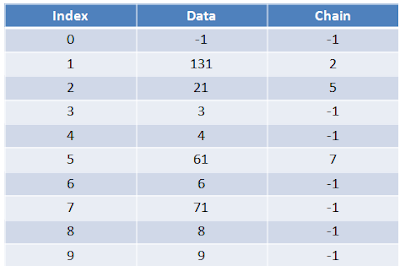
* In this method we will store collision element in next available space.
* Here when synonym is identified table is searched forward until an empty slot is found.
* The record is then inserted into empty slot.
* Number of probes in insert=number of probes in search

**Linear probing with Chaining=**

Here we will use hash value as an index to the array & one more field pointer. Each pointer accesses a chain of elements that share same hash location.In chaining without replacement method we maintain list of items which collide with an existing items. That is colliding records are chained to other locations.When we want to search first check that item at original location. Then check through chaining.

EX.Without replacement

131, 3, 4, 21, 61, 6, 71, 8, 9



**Chaining with Replacement=**

* To avoid difficulties in previous method chaining with replacement method is used.
* When original value comes that is placed at proper location.
* In the previous example if suppose original value comes then it is kept at next available location.
* But in chaining with replacement if u want to insert 15004 then it’s proper place is location 4. so this value is kept at proper place and 18003 is shifted to next available location.