Chapter 14 Hashing

What is hashing?

Used for storing and retrieving information as quickly as possible. Helpful in implementation of symbol tables.

Why hashing?

Because of its average best time complexity is O(1)

Hashtable ADT

Operations of the hast table are as follows:

- CreateHashTable
- HashSearch
- HashInsert
- HashDelete
- DeleteHashTable

Understanding Hashing

The process of mapping keys to memory locations is called hashing. We cannot use arrays instead of hashing as having an infinite possibilities of keys cannot have infinite memory locations, this is where hashing comes in.

Components of Hashing

- Hash Table: A data structure that stores the keys and their associated values.
- Hash Function: Function that converts key to a unique hash.
- Load factor = Number of elements in the hash table / Hash table size
- Collisions: When two or more keys generate the same hash that results in collision, a good hash function has in place efficient collision resolution techniques.
- Collision Resolution Techniques: The process of finding an alternate location is called collision resolution.
  - Direct chaining: An array of linked list application.
    - Separate chaining: Agar ek se jyada records same hash generate karte hai to un records ko singly linked mai daldo and us linked list ko attach card hashing key k sath, Matlab key will be associated with a singly linked list and this list will have the multiple values .
  - Open Addressing: Array-based implementation.
    - Linear probing: It has problems of clustering i.e. one part of hash table becomes dense and other part becomes sparse, due to clustering the searches are long thus affecting the overall efficiency of hashing.
      - rehash(key) = (n+1) % tablesize

        - Quadratic probing: The problem of clustering is eliminated using this method. If a location is occupied we use (i + 1^2), (i + 2^2), (i + 3^2), (i + 4^2) and so on.
            - rehash(key) = (n+k^2) % tablesize
        - Double hashing:
            - h2(key) != 0 & h2 != h1
            - We first probe the location *h*1(*key*). If the location is occupied, we probe the location *h*1(*key*) + *h*2(*key*), *h*1(*key*) + 2 * *h*2(*key*), ...

How hashing gets O(1) Complexity?

- The answer to this problem is simple. By using the load factor we make sure that each block (for example, linked list in separate chaining approach) on the average stores the maximum number of elements less than the *load factor*.
- If the average number of elements in a block is greater than the load factor, we rehash the elements with a bigger hash table size. One thing we should remember is that we consider average occupancy (total number of elements in the hash table divided by table size) when deciding the rehash.
- The access time of the table depends on the load factor which in turn depends on the hash function. This is because hash function distributes the elements to the hash table. For this reason, we say hash table gives O(1) complexity on average.

Bloom Filters