Hashing

Hashing is the process of mapping large amount of data item to smaller table with the help of hashing function. Hashing is also known as Hashing Algorithm or Message Digest Function.It is a technique to convert a range of key values into a range of indexes of an array. It is used to facilitate the next level searching method when compared with the linear or binary search.

Hashing allows to update and retrieve any data entry in a constant time O(1).Constant time O(1) means the operation does not depend on the size of the data. Hashing is used with a database to enable items to be retrieved more quickly. It is used in the encryption and decryption of digital signatures.

*****What is Hash Function?**

- A fixed process converts a key to a hash key is known as a Hash Function.
- This function takes a key and maps it to a value of a certain length which is called a Hash value or Hash.
- Hash value represents the original string of characters, but it is normally smaller than the original.
- It transfers the digital signature and then both hash value and signature are sent to the receiver. Receiver uses the same hash function to generate the hash value and then compares it to that received with the message.
- If the hash values are same, the message is transmitted without errors.

What is Hash Table?

- Hash table or hash map is a data structure used to store key-value pairs.
- It is a collection of items stored to make it easy to find them later.
- It uses a hash function to compute an index into an array of buckets or slots from which the desired value can be found.
- It is an array of list where each list is known as bucket.
- It contains value based on the key.
- Hash table is used to implement the map interface and extends Dictionary class.
- Hash table is synchronized and contains only unique elements.

Fig. Hash Table

- The above figure shows the hash table with the size of n = 10. Each position of the hash table is called as Slot. In the above hash table, there are n slots in the table, names = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}. Slot 0, slot 1, slot 2 and so on. Hash table contains no items, so every slot is empty.
- As we know the mapping between an item and the slot where item belongs in the hash table is called the hash function. The hash function takes any item in the collection and returns an integer in the range of slot names between 0 to n-1.

- Suppose we have integer items {26, 70, 18, 31, 54, 93}. One common method of determining a hash key is the division method of hashing and the formula is :

  Hash Key = Key Value % Number of Slots in the Table
- Division method or reminder method takes an item and divides it by the table size and returns the remainder as its hash value.

***Open Addressing

Like separate chaining, open addressing is a method for handling collisions. In Open Addressing, all elements are stored in the hash table itself. So at any point, size of the table must be greater than or equal to the total number of keys (Note that we can increase table size by copying old data if needed).
Insert(k): Keep probing until an empty slot is found. Once an empty slot is found, insert k
Search(k): Keep probing until slot's key doesn't become equal to k or an empty slot is reached.
Delete(k): Delete operation is interesting. If we simply delete a key, then search may fail. So slots of deleted keys are marked specially as "deleted".
Insert can insert an item in a deleted slot, but the search doesn't stop at a deleted slot.

***Comparison of all:

Linear probing has the best cache performance but suffers from clustering. One more advantage of Linear probing is easy to compute.
Quadratic probing lies between the two in terms of cache performance and clustering.
Double hashing has poor cache performance but no clustering. Double hashing requires more computation time as two hash functions need to be computed.

S.No.	Separate Chaining	Open Addressing
1.	Chaining is Simpler to implement.	Open Addressing requires more computation.
2.	In chaining, Hash table never fills up, we can always add more elements to chain.	In open addressing, table may become full.
3.	Chaining is Less sensitive to the hash function or load factors.	Open addressing requires extra care for to avoid clustering and load factor.
4.	Chaining is mostly used when it is unknown how many and how frequently keys may be inserted or deleted.	Open addressing is used when the frequency and number of keys is known.
5.	Cache performance of chaining is not	Open addressing provides better cache

S.No.	Separate Chaining	Open Addressing
	good as keys are stored using linked list.	performance as everything is stored in the same table.
6.	Wastage of Space (Some Parts of hash table in chaining are never used).	In Open addressing, a slot can be used even if an input doesn't map to it.
7.	Chaining uses extra space for links.	No links in Open addressing

****Coalesced hashing**

Coalesced hashing is a collision avoidance technique when there is a fixed sized data. It is a combination of both separate chaining and open addressing. It uses the concept of Open Addressing (linear probing) to find first empty place for colliding element from the bottom of the hash table and the concept of Separate Chaining to link the colliding elements to each other through pointers. The hash function used is h= (key)%(total number of keys). Inside the hash table, each node has three fields:

- h(key): The value of hash function for a key.
- Data: The key itself.
- Next: The link to the next colliding elements.

The basic operations of Coalesced hashing are:

1. INSERT(key): The insert Operation inserts the key according to the hash value of that key if that hash value in the table is empty otherwise the key is inserted in first empty place from the bottom of the hash table and the address of this empty place is mapped in NEXT field of the previous pointing node of the chain.(Explained in example below).
2. DELETE(Key): The key if present is deleted.Also if the node to be deleted contains the address of another node in hash table then this address is mapped in the NEXT field of the node pointing to the node which is to be deleted
3. SEARCH(key): Returns True if key is present, otherwise return False.

The best case complexity of all these operations is O(1) and the worst case complexity is O(n) where n is the total number of keys.It is better than separate chaining because it inserts the colliding element in the memory of hash table only instead of creating a new linked list as in separate chaining.

A perfect hash function is one that maps the set of actual key values to the table without any collisions. A minimal perfect hash function does so using a table that has only as many slots as there are key values to be hashed. If the set of keys IS known in advance, it is possible to construct a specialized hash function that is perfect, perhaps even minimal perfect. Algorithms for constructing perfect hash functions tend to be tedious, but a number are known,

This is used primarily when it is necessary to hash a relatively small collection of keys, such as the set of reserved words for a programming language.

The basic formula is:

h(S) = S.length() + g(S[0]) + g(S[S.length()-1])
where g() is constructed using Cichelli's algorithm so that h() will return a different hash value for each word in the set.
The algorithm has three phases: - computation of the letter frequencies in the words - ordering the words - searching