Hash Functions and Hash Tables

What is Problem Solving

- Hash Functions
- Different type of hash functions
- Collision Resolution
- Linear Probing
- Quadratic Probing
- Double Hashing
- Inserting and Deleting an element from a hash table

Hash Functions

Hashing is an data structure designed to solve the problem of efficiently finding and storing data in an array.

increases the efficiency of retrieval and optimises the search.

is a technique of mapping a large chunk of data into small tables using a hashing function.

message digest function.

technique that uniquely identifies a specific item from a collection of similar items.

uses hash tables to store the data in an array format.

Each value in the array has been assigned a unique index number.

Hash tables use a technique to generate these unique index numbers for each value stored in an array format.

This technique is called the hash technique.

Searching, insertion, removal operations will be easy with indexing

Hash techniques of search are performed on a part of the item i.e. key.

Each key has been mapped to a number, the range remains from 0 to table size 1

Hash Function

Hash function converts the item into a small integer or hash value.

used as an index to store the original data.

It stores the data in a hash table. You can use a hash key to locate data quickly.

Ex-

Room No. in a building

The hash function maps the arbitrary size of data to fixed-sized data.

It returns the following values: a small integer value (also known as hash value), hash codes, and hash sums.

Hash function= key % array_size⇒hash value⇒bucket id/index

index = key % array_size

The hash function must satisfy the following requirements:

1. easy to compute.
2. never gets stuck in clustering and distributes keys evenly across the hash table.
3. avoids collision when two elements or items get assigned to the same hash value.

Hash Table

Hashing in data structure uses hash tables to store the key-value pairs.

The hash table then uses the hash function to generate an index.

Hashing uses this unique index to perform insert, update, and search operations.

It can be defined as a bucket where the data are stored in an array format. These data have their own index value. If the index values are known then the process of accessing the data is quicker.

example. Store items (arranged in a key-value pair) inside a hash table with 30 cells.

The values are: (3,21) (1,72) (40,36) (5,30) (11,44) (15,33) (18,12) (16,80) (38,99)

Serial No.	Key	Hash	Array Index
1	3	3%30 = 3	3
2	1	1%30 = 1	1
3	40	40%30 = 10	10
4	5	5%30 = 5	5
5	11	11%30 = 11	11
--			
6	38	38%30=8	8

Collision Resolution Techniques

Hashing in data structure falls into a collision if two keys are assigned the same index number in the hash table.

The collision creates a problem because each index in a hash table is supposed to store only one value.

To avoid collision hashing uses some techniques

It is a process of finding an alternate location.

The collision resolution techniques can be named as-

Open Hashing (Separate Chaining)

Closed Hashing (Open Addressing)

- Linear Probing
- Quadratic Probing
- Double Hashing

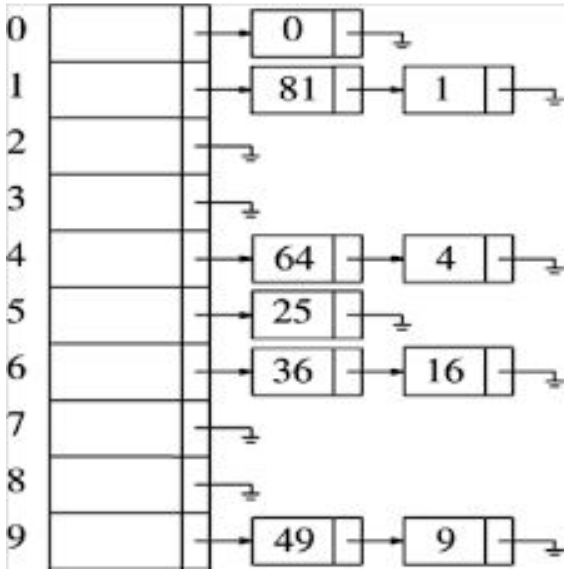
Open Hashing (Separate Chaining)

To resolve collision at every index of bucket list will be maintained (SLL)

Each new elt with same bucket id will be added in list at beginning/end.

Array will be containing nodes will

maintain list



Close Hashing (Open Addressing)

Linear Probing

A sequential search can be performed to find new index if any collision occurred.

index that is already occupied and new value got the same index with hash function -situation of collision

To resolve collision in linear probing hashing performs a search operation and probes linearly for the next empty cell.

h(k)=k%n

h’(k)=h(k)+i —> i is prob value

easiest way to resolve any collisions in hash tables.

In linear probing we need to remember that prob value the no. of times we checked for new place.

Adv -

No Extra space

Dis Adv-

Searching is not that efficient - Time Comp. -O(1) / Worst Case - O(n)

Deletion is problematic - as if any elt deleted and space is vacant then search can be stopped due to empty block as no elt is there. Even though elt is present

So need to take action for deletion like fill empty space with some keyword

Primary clustering - prob. Of any index will be increased coZ of group of cluster

Searching in prim. Clustoring will be time consuming

Imagine you have been asked to store some items inside a hash table of size 30. The items are already sorted in a key-value pair format. The values given are: (3,21) (1,72) (63,36) (5,30) (11,44) (15,33) (18,12) (16,80) (46,99).

The hash(n) is the index computed using a hash function and T is the table size.

If slot index = (hash(n) % T) is full, then we look for the next slot index by adding 1 ((hash(n) + 1) % T).

If (hash(n) + 1) % T is also full,

then we try (hash(n) + 2) % T.

If (hash(n) + 2) % T is also full, then we try (hash(n) + 3) % T

Number	Key	Hash	Array Index	Array Index after Linear Probing
1	3	3%30 = 3	3	3
2	1	1%30 = 1	1	1
3	63	63%30 = 3	3	4
4	5	5%30 = 5	5	5
5	11	11%30 = 11	11	11
6	15	15%30 = 15	15	15
7	18	18%30 = 18	18	18
8	16	16%30 = 16	16	16
9	46	46%30 = 16	16	17

Quadratic Probing

Quadratic probing is an open-addressing scheme where we look for the i2‘th slot in the i’th iteration if the given hash value x collides in the hash table.

How Quadratic Probing is done?
h(k)=k%size
h’(k)=(h(k)+i^2) /% 10

Adv
No Extra Space
Primary Clustering

Dis Adv-
 No guarantee of finding place/slot if HT is half full
Search- o(n)
Example -Imagine you need to store some items inside a hash table of size 20. The values given are: (16, 8, 63, 9, 27, 37, 48, 5, 69, 34, 1).

h(n)=n%20   if collision h’(n,i)= (i^2+h(n))%20

N	i	h(n,i) = (h(n) + i2) %20
16	i = 0,	h(n,0) = 16
8	i = 0,	h(n,0) = 8
63	i = 0,	h(n,0) = 3
9	i = 0,	h(n,0) = 9
27	i = 0,	h(n,0) = 7
37	i = 0,	h(n,0) = 17
48	i = 0,	h(n,0) = 8
	i = 1,	h(n,1) = 8+1=9
	i = 2,	h(n,2) = 8+4=12
5	i = 0, h(n,0) = 5	
69	i = 0, h(n,0) = 9	
	i = 0, h(n,1) = 9+1=10	
34	i = 0, h(n,0) = 14	
1	i = 0, h(n,0) = 1	

Double Hashing

The double hashing technique uses two hash functions.

The second hash function comes into use when the first function causes a collision.

It provides an offset index to store the value.

The formula for the double hashing technique is as follows:

(firstHash(key) + i * secondHash(key)) % sizeOfTable

Where i is the offset value. This offset value keeps incremented until it finds an empty slot.

For example, you have two hash functions: h1 and h2. You must perform the following steps to find an empty slot:

1. Verify if hash1(key) is empty. If yes, then store the value on this slot.
2. If hash1(key) is not empty, then find another slot using hash2(key).
3. Verify if hash1(key) + hash2(key) is empty. If yes, then store the value on this slot.
4. Keep incrementing the counter and repeat with hash1(key)+2hash2(key), hash1(key)+3hash2(key), and so on, until it finds an empty slot.

Note - if addition of phishing greater than size of array then do mod

Double Hashing Example

Imagine you need to store some items inside a hash table of size 20. The values given are: (16, 8, 63, 9, 27, 37, 48, 5, 69, 34, 1).

h1(n)=n%20
h2(n)=n%13
n h(n, i) = (h1 (n) + ih2(n)) mod 20

h(n)=n%20 if collision h'(n,i)= (h1(n)+h2(n))%20

N	i	h(n,i) = h(n) %20 / (h1(n)+i*h2(n))%20
16	i = 0,	h(n,0) = 16
8	i = 0,	h(n,0) = 8
63	i = 0,	h(n,0) = 3
9	i = 0,	h(n,0) = 9
27	i = 0,	h(n,0) = 7
37	i = 0,	h(n,0) = 17
48	i = 0,	h(n,0) = 8
	i = 1,	h(n,1) = 8+9=17
	i = 2,	h(n,2) = 8+18=26%20=6
5	i = 0,	h(n,0) = 5
69	i = 0,	h(n,0) = 9
	i = 0,	h(n,1) = 9+9=18
34	i = 0,	h(n,0) = 14
1	i = 0,	h(n,0) = 1