Hashing

Hashing is a technique used for performing insertions, deletions and finds in constant average time.

The implementation of hash tables is frequently called hashing.

Hash table:-

. It is a non-linear data structure used to store vectors or pair of (key,value) into a hash table.

→ Storing and retreiving the records into hash table. Performed hash on a special value called key.

→ Hash table data structure is merely an array of some fixed size containing keys.

→ A key is a string with an associate value.

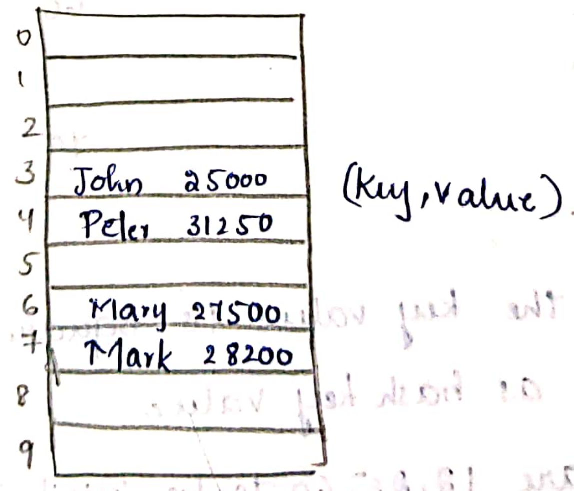
→ We refer table size as Tablesize and the hashtable will run from 0 to Tablesize-1

→ Each key is mapped into some number in the range 0 to Tablesize-1 and placed in the appropriate cell.

→ The mapping is called hash function which is ideally should be simple to compute and should ensure that two distinct or keys get different cell

→ Hash function returns a value called hash key.

⇒ Based on key values records are stored and retrieved.



Ideal hash table

Hash function:-

It is a function that returns hash key. Based on this key the records are inserted, deleted and searched. Hash function accepts key as input and produces hash key as output.

Types of hash functions:

1. Division method
2. Mid square method
3. Digit folding method (or) pairing method
4. Multiplicative method

1. Division Method

It is the simplest method of hash function. One hash function depends upon the remainder value of division operation.

One remainder value is treated as hash key value.

H(key) = key % table size

Here key is a record unique value and table size is nothing but hashtable size

Eg: Let keys (records) are 55, 66, 72, 83, 79.

H(key) = key % table size

H(55) = 55 % 10 = 5

H(66) = 66 % 10 = 6

H(72) = 72 % 10 = 2

H(83) = 83 % 10 = 3

H(79) = 79 % 10 = 9

0	
1	
2	72
3	83
4	
5	55
6	66
7	
8	
9	79

2. Mid Square Method :-

In this method, the key value is squared and the middle po
of the result is used as hash key value.

Let keys (records) are 12, 25, 60 to be inserted in hashtable and le
table size is 10.

$H(12) = 12^2 = 1\underline{4}4 = 4$

$H(25) = 25^2 = 6\underline{2}5 = 2$

$H(60) = 60^2 = 36\underline{0}0 = 0$

0	60
1	
2	25
3	
4	12
5	
6	
7	
8	
9	

3. <u>Digit folding method</u> (or) <u>pairing method</u> :-

In this method, the key is divided into separate parts and using some simple operations these parts are combined to produce hash key value.

Let key(record) is 12345 & table size is 200

Eg: One key is divided into 3 parts and perform addition operation

```
 12
 34
  5
----
 51
```

So, the record 12345 is stored at 51th location.

Eg:
Let key(record) is 5678 & table size is 200

One key is divided into 2 parts and perform addition operation

```
  56
+ 78
-----
 134
```

So, the record 5678 is stored at 134th location of hash-table

4. <u>Multiplicative method</u> :-

To find the key, we use the following formula.

$H(key) = floor(table size * (key * A)) \bmod table size$

Donald Knuth suggested to used constant

$A = 0.618033$

Let table size is 10

Key is 23

$H(23) = floor(10 * (23 * 0.618033)) \bmod 10$

$= floor(142.14) \bmod 10$

= 142 % 10

= 2

So key 23 is stored at location 2 of hash table.

Characteristics of a good hash function:

1. The has-function should be simple to compute.

2. Number of collisions should be less while placing the key in hash table.

3. The cost of computing a hash function should be less.

4. Hash function should produce hash key which will be distributed uniformly across the hash table.

5. The hash function should make use of all information provided by the key.

Collision :-

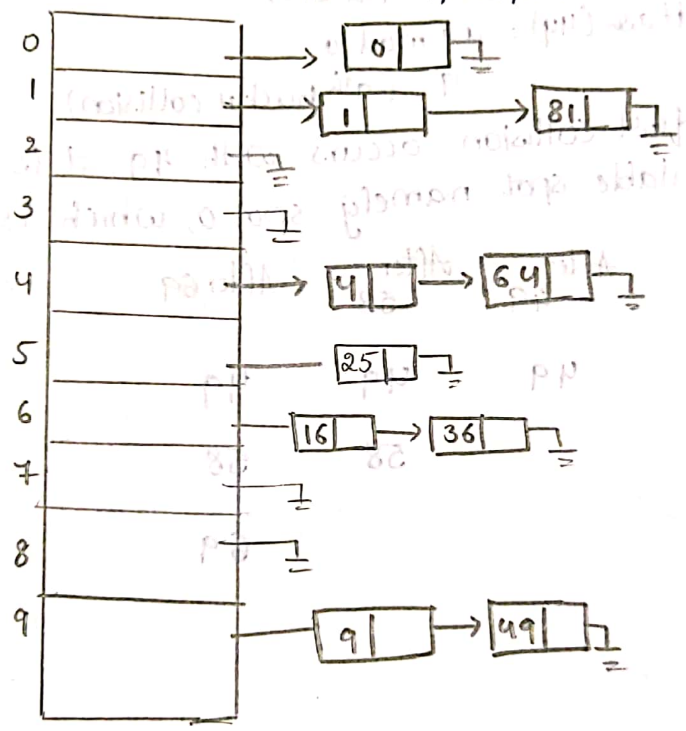
→ When an element is inserted if it hashes to same bucket which is already having an element then we have a collision and need to resolve it.

There are several methods for dealing collision resolution techniques. Separate chaining and open addressing.

Separate Chaining:- The first strategy commonly a separate chaining. Separate chaining is to keep a linked list of all elements that hash to same value.

Hash(x) = x mod 10

Keys = 0, 1, 4, 9, 16, 25, 36, 49, 64, 81



Separate chaining hash table

Load factor (λ):- load factor of a hash table is the ratio of the nu of elements in hash table to the table size

$$\lambda = \frac{\text{no. of keys in hashtable}}{\text{tablesize}}$$

Loadfactor should be less and it is important.

In separate chaining λ is 1 and table size should be prime.

Linear Probing:-

This is the easiest method of collision resolution techniques.

In linear probing F, collision resolution strategy is linear

$F(i) = i$

This amounts to trying cells sequentially. in search of empty cell.

For example inserting keys in to hash table {89, 18, 49, 58, 69} using the hash function and collision resolution strategy is $F(i) = i$

	Empty table
0	
1	
2	
3	
4	
5	
6	
7	
8	
9	

Hash (89) = 89 mod 10
= 9 (9th bucket)

Hash (18) = 18 mod 10
= 8 (8th bucket)

Hash (49) = 49 mod 10
= 9 (9th bucket collision)

The first collision occurs with 49, it is put in the next available spot namely spot 0, which is open.

	Empty table	After 89	After 18	After 49	After 58	After 69
0				49	49	49
1					58	58
2						69
3						
4						
5						
6						
7						
8			18	18	18	18
9		89	89	89	89	89

→ When a collision occurs it can be resolved by placing the secon key linearly down whenever empty bucket is found.

In case of 49 next bucket empty is bucket 0

Hash (58) = 58 mod 10 = 8 second collision

The key 58 collides with 18, 89 and 49.
Before an empty cell (or) bucket is found three buckets away

$$Hash(69) = 69 \bmod 10 = 9$$

For 69 it is placed in bucket after searching linearly empty bucket is found at 2.

Drawbacks:

1. As long as table is big enough empty bucket can always be found but the time to do so is quite large
2. When the hash is relative empty (i.e) only one or two buckets are empty blocks of occupied cells or buckets are forming. This effect is know as primary clustering i.e that any key that hashes into cluster will requir several attempts to resolve the collision and then it will add to the cluster.

For insertion and unsuccessful searches no. of probes is $\frac{1}{2}\left(1+\frac{1}{(1-\lambda)^2}\right)$

Successful searches no. of probes is $\frac{1}{2}\left(1+\frac{1}{1+\lambda}\right)$

λ is load factor.

Quadratic Probing:-

The primary clustering problem of linear probing can be solved by Quadratic Probing.

In Quadratic Probing collision resolution function is quadratic i.e., F(

$$keys = \{89, 18, 49, 58, 69\}$$

$$Hash(x) = x \bmod tablesize, \quad tablesize = 10$$

$$Hash(89) = 89 \bmod 10$$
$$= 9$$
$$Hash(18) = 18 \bmod 10$$
$$= 8$$
$$Hash(49) = 49 \bmod 10$$
$$= 9$$

	Empty table	After 89	After 18	After 49	58	69
0				49	49	49
1					58	58
2						69
3						
4						
5						
6						
7						
8			18	18	18	18
9		89	89	89	89	89

When 49 collides with 89, the next position attempted is one bucket away. Therefore 49 is placed in bucket 0 which is empty.

$$H(58) = 58 \% 10$$
$$= 8$$

58 collides at bucket 8 then the cell or bucket one away is tried, but another collision occurs.

A vacant cell is found at the next cell is tried which is $2^2 = 4$ away.

58 is placed at bucket 2.

$$H(69) = 69 \% 10$$
$$= 9$$

69 Collides with 89 next buck is tried which results another collision so next bucket tried

$F(i) = i^2 = 2^2 = 4$ buckets away.

Drawbacks:-

For linear probing it is bad idea when hash table gets nearly full because performance degrades.

For Quadratic probing, the situation even more as there is no gua of finding an empty cell once the hash-table gets half full if hash table is not prime

→ If the table is even one more than half-full insertion could fail.

⇒ Secondary clustering occurs.

Quadratic probing does not require second hash function and it is simpler and faster.

Double Hashing:-

The last collision resolution method is double hashing.

For double hashing, collision resolution function $F(i) = i \cdot hash_2(X)$

This formula says that we apply a second hash function to X and probe at a distance $hash_2(x)$, $2hash_2(x)$ ... so on

A second hash function $hash_2(x) = R - (X \bmod R)$, with R a prime smaller than Table size.

If we choose R=7.

89, 18, 49, 58, 69

$hash_1(89) = 89 \% 10 = 9$

$hash_1(18) = 18 \% 10 = 8$

$hash_1(49) = 49 \% 10 = 9$ collision occurs

Second $hash_2(49) = R - (X \bmod R)$

$= 7 - (49 \bmod 7)$

$= 7 - 0 = 7$ insert in position 6.

	Empty Table	After 89	After 18	After 49	After 58	After 69
0						69
1						
2						
3					58	58
4						
5						
6					49	49
7						
8			18	18	18	18
9		89		89	89	89

$hash_1(58) = 58 \bmod 10 = 8$ collision occurs

$hash_2(58) = 7 - (58 \% 7)$

$= 7 - 2 = 5$

$hash_1(69) = 69 \bmod 10 = 9$ collision occurs

$hash_2(69) = 7 - (69 \bmod 7)$

$= 7 - 6$

$= 1$

For double hashing the Table size has to be prime.

For example to insert 23.

Rehashing:

If the table gets too full, the running time for the operations start taking too long and inserts might fail for open addressing hashing with quadratic resolution. This can happen if there are too many removals intermixed with insertions.

A solution, is to build another table that is about twice as big with an associated new hash function and scan down, the entire original hash table, Computing new hash value for each key and inserting it in the new table.

13, 15, 26 and 6 are inserted into an open addressing hash table size of 7

$hash(x) = X \bmod 7 = 13 \bmod 7 = 6$

$hash(15) = 15 \bmod 7 = 1$

$hash(24) = 24 \bmod 7 = 3$

$hash(6) = 6 \bmod 7 = 6$

0	6
1	15
2	
3	24
4	
5	
6	13

Open addressing hashtable with linear probing
13, 15, 6, 24

0	6
1	15
2	23
3	24
4	
5	
6	13

After inserting
$hash(23) = 23 \bmod 7$
$= 2$

→ After 23 is inserted into table the resulting table is 70% full.

→ Because the table is so full, a new table is created

Table size = 17

New hash function $h(X) = X \bmod 17$

The old table is scanned and keys 6,15,23,24 and 13 are inserted into newtable.

→ This entire operation is rehashing.

→ This expensive operation requires O(N) since there are N elements to rehash.

0	
1	
2	
3	
4	
5	23
6	6
7	24
8	
9	
10	
11	
12	
13	13
14	
15	15
16	

$h(6) = 6 \bmod 17 = 6$

$h(15) = 15 \bmod 17 = 15$

$h(23) = 23 \bmod 17 = 5$

$h(24) = 24 \bmod 17 = 7$

$h(13) = 13 \bmod 17 = 13$

Open addressing hash table after rehashing

→ Rehashing as soon as table is half full.

→ Rehashing can also be done when insertion fails.

→ Rehashing can be done when load factor (λ) reached certain value.

Scanned with CamScanner