UNIT-10 HASHING

Introduction: -

Why Hashing?

Internet has grown to millions of users generating terabytes of content every day. According to internet data tracking services, the amount of content on the internet doubles every six months.

With this kind of growth, it is impossible to find anything in the internet, unless we develop new data structures and algorithms for storing and accessing data. So, what is wrong with traditional data structures like Arrays and Linked Lists?

Suppose we have a very large data set stored in an array. If the array is sorted then a technique such as binary search can be used to search the array. Otherwise, the array must be searched linearly. Either case may not be desirable if we need to process a very large data set.

So, we need a search algorithm which performs search in a constant time. Ideally it is impossible to achieve a performance of constant time, but a search algorithm can be derived which gives performance very close to O (1). This search algorithm is called as **Hashing**.

Hashing: -

Hashing uses a data structure which is known as Hash Table. Hash table is a fixed size array in which the items are inserted using Hash functions.

Hash function: -

A Hash function is simply a mathematical function which maps the key to some specific slot in the hash table. So, we need to apply the hash function on the key, then we get the positional index of the hash table on which the new item is going to be insert or item can be searched out.

So,

Hashing – is a technique or an algorithm.

Hash tables – Fixed size array

Hash function – Apply on keys to get hash Values (Index of hash Table)

Collision – The situation in which a key hash to on same index which is already occupied by another key. That means when we get the hash value of the given key same as the index occupied by another key is called as collision.

Collision Resolution – The process to finding another location for the given key value is called as collision resolution.

Hash Functions: -

Division-Remainder Method: -

- Also called as Simplest Method
- Most commonly used method
- Given values: - Key = K and Number of slots in hash table = N
- K is divided by N and the remainder become the index of hash table
- **h(k) = k mod N**
- Index of hash table = k % N.
- Technique is very well if the N is either a prime number not to close to a power of two.
- Ex: N = 101 , Key = 1123456 So Answer of Index = Key mod 101 => 1123456 % 101 = 33.

So the key K is stored on 33 index of hash table.

h(1123456) = 1123456 mod 101 = 33.

Collision: -

- Hash function gets us a small number for a key which is a big integer or string, there is a possibility that two keys result in the same value.

- The situation where a newly inserted key map to an already occupied slot in the hash table is called collision
- and It must be handled using some collision handling technique.

Example: -

h(21) = 21 % 11 = 10

h(56) = 56 % 11 = 1

h(78) = 78 % 11 = 1

- h(K) = K mod N
- Ex : Insert values in Hash tables where N = 11
- Values : 21, 56,78,133,121,65

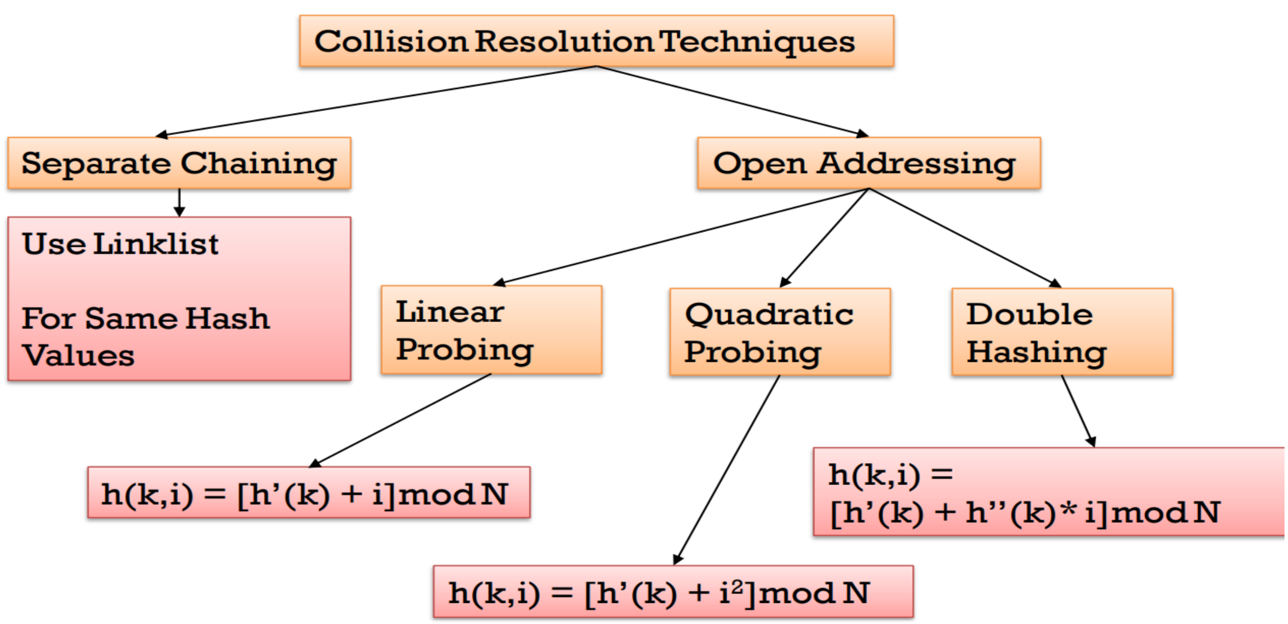
0	1	2	3	4	5	6	7	8	9	10
	56									**21**

78

h(78) = 78 % 11 = 1 → Can Not insert at same Position

So, this is known as Collision.... So, we need collision resolution methods to solve this issue.

Collision Resolution Techniques: -



1. Separate Chaining: -

Collisions are resolved using a list of elements to store objects with the same key together.

Ex : 20, 32, 41, 66, 72, 50, 106, 51, 71

h(k) = k mod N

N = 10

h(20) = 20 mod 10 = 0

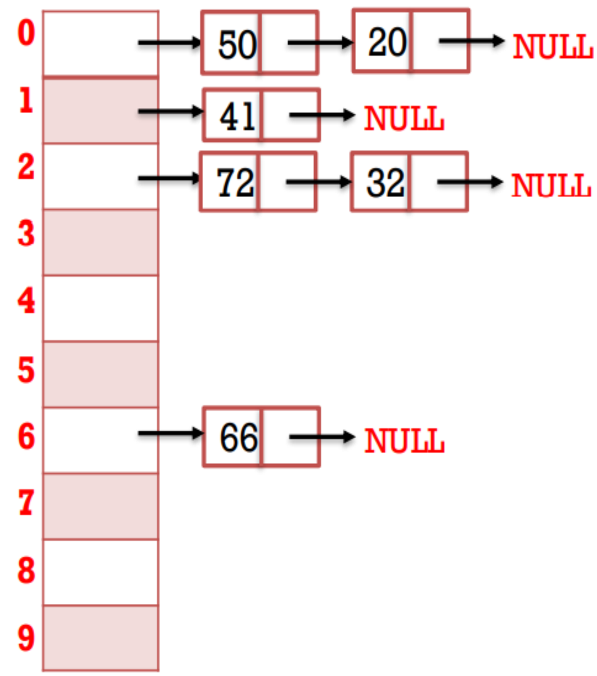
h(32) = 32 mod 10 = 2

h(41) = 41 mod 10 = 1

h(66) = 66 mod 10 = 6

h(72) = 72 mod 10 = 2

h(50) = 50 mod 10 = 0



Advantages: -

It is easy to implement.

The hash table never fills full, so we can add more elements to the chain.

It is less sensitive to the function of the hashing.

Disadvantages: -

In this, cache performance of chaining is not good.

The memory wastage is too much in this method.

It requires more space for element links.

2. Open Addressing: -

• Instead of in linked lists, all entry records are stored in the array itself.

• When a new entry has to be inserted, the hash index of the hashed value is computed and then the array is examined (starting with the hashed index).

• If the slot at the hashed index is unoccupied, then the entry record is inserted in slot at the hashed index else it proceeds in some probe sequence until it finds an unoccupied slot.

(i) Linear Probing: -

The idea of linear probing is simple, we take a fixed sized hash table and every time we face a hash collision, we linearly traverse the table in a cyclic manner to find the next empty slot.

Formula: h(k, i) = [h'(k) + i] mod n, h'(k) = Any Hash Function

Ex : 176, 75, 37, 56, 79, 154, 10, 99 N = 10

h(176, 0) = [176 mod 10 + 0] mod 10

= [6 + 0] mod 10

= [6] mod 10

= 6

h(75, 0) = [75 mod 10 + 0] mod 10

= [5 + 0] mod 10

= [5] mod 10

=5

h(37, 0) = [37 mod 10 + 0] mod 10

= [7 + 0] mod 10

= [7] mod 10

=7

0	1	2	3	4	5	6	7	8	9
					75	176	37		

h(56, 0) = [56 mod 10 + 0] mod 10

= [6 + 0] mod 10

= [6] mod 10

= 6

6 – Already Occupied So Take i = 1

h(56, 1) = [56 mod 10 + 1] mod 10

= [6 + 1] mod 10

= [7] mod 10

= 7

7 – Already Occupied So Take i = 2

h(56, 2) = [56 mod 10 + 2] mod 10

= [6 + 2] mod 10

= [8] mod 10

= 8

0	1	2	3	4	5	6	7	8	9
					75	176	37	56	

h(79, 0) = [79 mod 10 + 0] mod 10

= [9 + 0] mod 10

= [9] mod 10

= 9

h(154, 0) = [154 mod 10 + 0] mod 10

= [4 + 0] mod 10

= [4] mod 10

= 4

0	1	2	3	4	5	6	7	8	9
				154	75	176	37	56	79

h(10, 0) = [10 mod 10 + 0] mod 10

= [0 + 0] mod 10

= [0] mod 10

= 0

h(99, 0) = [99 mod 10 + 0] mod 10

= [9 + 0] mod 10

= [9] mod 10

= 9

9 – Already Occupied So Take i = 1

h(99, 1) = [99 mod 10 + 1] mod 10

= [9 + 1] mod 10

= [10] mod 10

= 0

9 – Already Occupied So Take i = 2

h(99, 1) = [99 mod 10 + 2] mod 10

= [9 + 2] mod 10

= [11] mod 10

= 1

0	1	2	3	4	5	6	7	8	9
10	99			154	75	176	37	56	79

Disadvantages: -

The main problem is clustering.

It takes too much time to find an empty slot.

(ii) Quadratic Probing: -

Formula: h (k, i) = [h'(k) + i^2] mod n, h'(k) = Any Hash Function

Ex : 126, 75, 37, 56, 29, 154, 10, 99 N = 11

h(126, 0) = [126 mod 11 + 0^2] mod 11

= [5 + 0] mod 11

= [5] mod 11

= 5

h(75, 0) = [75 mod 11 + 0^2] mod 11

= [9 + 0] mod 11

= [9] mod 11

= 9

h(37, 0) = [37 mod 11 + 0^2] mod 11

= [4 + 0] mod 11

= [4] mod 11

= 4

h(56, 0) = [56 mod 11 + 0^2] mod 11

= [1 + 0] mod 11

= [1] mod 11

= 1

h(29, 0) = [29 mod 11 + 0^2] mod 11

= [7 + 0] mod 11

= [7] mod 11

= 7

h(154, 0) = [154 mod 11 + 0^2] mod 11

= [0 + 0] mod 11

= [0] mod 11

= 0

h(10, 0) = [10 mod 11 + 0^2] mod 11

= [10 + 0] mod 11

= [10] mod 11

= 10

0	1	2	3	4	5	6	7	8	9	10
154	**56**			**37**	**126**		**29**		**75**	**10**

h(99, 0) = [99 mod 11 + 0^2] mod 11

= [0 + 0] mod 11

= [0] mod 11

= 0

0 – Already Occupied So Take i = 1

h(99, 0) = [99 mod 11 + 1^2] mod 11

= [0 + 1] mod 11

= [1] mod 11

= 1

0	1	2	3	4	5	6	7	8	9	10
154	56			37	126		29		75	10

1 – Already Occupied So Take i = 2

h(99, 0) = [99 mod 11 + 2^2] mod 11

= [0 + 4] mod 11

= [4] mod 11

= 4

4 – Already Occupied So Take i = 3

h(99, 0) = [99 mod 11 + 3^2] mod 11

= [0 + 9] mod 11

= [9] mod 11

= 9

9 – Already Occupied So Take i = 4

h(99, 0) = [99 mod 11 + 4^2] mod 11

= [0 + 16] mod 11

= [16] mod 11

= 5

5 – Already Occupied So Take i = 5

h(99, 0) = [99 mod 11 + 5^2] mod 11

= [0 + 25] mod 11

= [25] mod 11

= 3

3 – Empty

0	1	2	3	4	5	6	7	8	9	10
154	56		99	37	126		29		75	10

(iii) Double Hashing: -

Formula: h(k, i) = [h'(k) + i*h''(k)] mod n

h'(k) = k mod N ➔ N = 13

h''(k) = k mod N' , N' = Prime value smaller than the Table Size ➔ N' = 11

Ex : 126, 75, 37, 56, 29, 152 N = 13 & N' = 11

h(126, 0) = [126 mod 13 + 0 * (126 mod 11)] mod 13

= [9 + 0] mod 13

= [9] mod 13

= 9

h(75, 0) = [75 mod 13 + 0 * (75 mod 11)] mod 13

= [10 + 0] mod 13

= [10] mod 13

= 10

h(37, 0) = [37 mod 13 + 0 * (37 mod 11)] mod 13

= [11 + 0] mod 13

= [11] mod 13

= 11

h(56, 0) = [56 mod 13 + 0 * (56 mod 11)] mod 13

= [4 + 0] mod 13

= [4] mod 13

= 4

h(29, 0) = [29 mod 13 + 0 * (29 mod 11)] mod 13

= [3 + 0] mod 13

= [3] mod 13

= 3

h(152, 0) = [152 mod 13 + 0 * (152 mod 11)] mod 13

= [9 + 0] mod 13

= [9] mod 13

= 9

Take i=1

h(152, 1) = [152 mod 13 + 1 * (152 mod 11)] mod 13

= [9 + 9] mod 13

= [18] mod 13

=5

0	1	2	3	4	5	6	7	8	9	10	11	12
		37	**29**	**56**	**152**				**126**	**75**	**37**	