

03-12-2020

# Hashing

# Searching

- *Searching are very common operations in most computer applications.*
- ***If we browse the Internet there is virtually no page where we will not found a search button!***

# Searching

- *The Google search itself is an example, which proves how search facility helps Internet users.*
- *Windows operating systems also have search facility to find files and folders.*
- *Thus searching is the term defined to retrieve a record with a particular value.*

# Searching

## *Different types of Searching :-*

- *The different types of searching techniques are*
  - *Linear (Sequential) Search*
  - *Binary Search*

# Searching

- **Linear (Sequential Search) :-**
- *Linear (Sequential) Search is a method where the search begins at one end of the list, scans the elements of the list , one by one until the desired record is found or the end of the array/list is reached.*
- **If Search is successful then it will return the location of element otherwise it will return the failure notification**

# Searching

- **Binary Search :-**
- *In Binary Search, the entire sorted list is divided into two parts.*
- *We first compare our input item with the mid element of the list and*
- ***Restrict our attention to only the first or second half of the list depending on whether the input item comes before or after the mid-element.***
- *In this way we reduce the length of the list at each step until we are getting a single element in a list.*

# Hashing

- Various search algorithms allows us to search data in linear time complexity
  - **$[O(n)] \Rightarrow \text{Linear Search}$**
  - **logarithmic time complexity  $[O(\log n)] \Rightarrow \text{Binary Search}$**
- To search a data in constant time complexity i.e. in  $O(1)$  time ??

# Hashing

- To search a data in constant time complexity i.e. in  $O(1)$  time ??
  - Hash tables
  - Allows the storage and retrieval of data in an average time of  $O(1)$ .



# Hashing

- Hashing is an important Data Structure which is designed to use a **special function called the Hash function** which is used to map a given value with a particular key for faster access of elements.
- **The efficiency of mapping depends of the efficiency of the hash function used.**
- It is a technique whereby items are placed into a structure based on a **key to-address transformation**.

# Hashing

**For storing record**

**Key**



**Generate array index**



**Store the record on that array index**

# Hashing

**For accessing record**

**Key**



**Generate array index**



**Get the record from that array index**

# Hashing

## Hash Table :-

- A hash table is a data structure that uses a **random access data structure, such as an array,** and a **mapping function, called a hash function,** to allow average constant time  **$O(1)$  searches.**

# Hashing

## Hash Function :-

- A hash function is a mapping between a set of input values and a set of integers, known as hash values.
- Denoted by H.

$$H(K) \rightarrow A$$

# Hashing

## Good Hash Function :-

- Should have following properties

- 1) Efficiently computable.
- 2) Should uniformly distribute the keys (Each table position equally likely for each key)
- 3) Should generate unique addresses or addresses with minimum collision

# Hashing

## Hash Function :-

- For eg - A function that converts a given big phone number to a small practical integer value. The mapped integer value is used as an index in hash table.
- In simple terms, **a hash function maps a big number or string to a small integer that can be used as index in hash table.**

# Hashing

## Hash Function :-

Array Index	Keys
0	
1	
2	011 Delhi
3	
4	022 Kolkata
5	
6	033 Mumbai
7	
8	044 Chennai
9	



# Hashing

## Hash Function :-

STD codes of the cities are keys

Hash function maps that into address by just adding the digits of the key

Array Index	Keys
0	
1	
2	011 Delhi
3	
4	022 Kolkata
5	
6	033 Mumbai
7	
8	044 Chennai
9	

Prof. Shweta Dhawan Chachra

# Hashing

## Hash of key :-

- Let  $h$  be a hash function &  $K$  is a key,
- $H(K)$  is called hash-of-key.
- The hash-of-key is the index at which a record with the key value  $K$  must be kept.**

# Hashing

If each key is mapped on a unique hash table address then this situation is **ideal situation**

If the hash function is generating same hash table address for different keys, **the situation is called collision**

A good hash function should **generate minimum collision**

# Hashing

**2 things-**

- 1) A good Choosing a hash function which ensures minimum collision**
- 2) Resolving collision**

# Hashing

**Some techniques for choosing hash functions are**  
**Truncation Method**

# Hashing

## Truncation Method

- Easiest method
- **A part of the key as address**
- **Can be rightmost or leftmost digit**

Eg-

82394561, 87139465, 83567271, 85943228

Suppose table size is 100 then take **the 2 rightmost digits for getting the addresses.**

Address will be 61, 65, 71 and 28

# Hashing

## Mid Square Method

- **Square** the key
- After getting the number, we take **some digits from the middle** of that no as address

Eg-

**1337 , 1273, 1391, 1026**

**Square=1787569, 1620529, 1934881, 1052676**

Lets take 3<sup>rd</sup>, 4<sup>th</sup> digit from each number as address

Let the table size be 100

**Address=75, 05,48, 26**

# Hashing

## Folding Method

- Break the key into pieces, add them and get the hash address

Eg-Lets take some 8 bit address

82394561, 87139465, 83567271, 85943228

Chop them in pieces 3,2 and 3 digits and them

**Address will be->**

$$82394561 = 823+94+561 = 1478$$

$$87139465 = 871+39+465 = 1375$$

$$83567271 = 835+67+271 = 1173$$

$$85943228 = 859+43+228 = 1130$$



# Hashing

## Folding Method

Suppose the size of Hash table=1000,

Hash address will be from 0 to 999

**Truncate the higher digits of the number**

**Address will be->**

**$H(82394561) = 478$**

**$H(87139465) = 375$**

**$H(83567271) = 173$**

**$H(85943228) = 130$**

# Hashing

## Modular Method

- Best way
- Perform Modulus operation, Remainder is address of hash table
- Ensure address will be in range of hash table
- Take table size as a prime number
- Table size should not be in Power of 2, else collisions increase

# Hashing

## Modular Method

- Let us take some keys
- 82394561, 87139465, 83567271, 85943228
- Table size=97

Address=

$$82394561 \% 97 = 45$$

$$87139465 \% 97 = 0$$

$$83567271 \% 97 = 25$$

$$85943228 \% 97 = 64$$

# Hashing

## **Closed Hashing( Open addressing)**

- In Open hashing, we use linked list for resolving collision resolution which has its own disadvantage
- Uses array
- Techniques used in Closed Hashing are-
  - 1) **Linear Probing**
  - 2) **Quadratic Probing**
  - 3) **Double Hashing**

# Hashing

## Linear Probing

- 1) Map the key on a particular address using Modular method
- 2) If 2 keys have the same hash address, then it will find the next empty position in the hash table
- 3) Hash table is used as circular array
- 4) If table size is  $N$  then after  $N-1$  position

# Hashing

## Linear Probing

Elements=29,18,43,10,36,25,46

Table size=11

# Hashing

## Linear Probing

Elements=29,18,43,10,36,25,46

Table size=11

$$H(29)=29\%11=7$$

$$H(18)=18\%11=7$$

$$H(43)=43\%11=10$$

$$H(10)=10\%11=10$$

$$H(36)=36\%11=3$$

$$H(25)=25\%11=3$$

$$H(46)=46\%11=2$$

# Hashing

## Linear Probing

$$H(29) = 29 \% 11 = 7$$

$$H(18) = 18 \% 11 = 7$$

$$H(43) = 43 \% 11 = 10$$

$$H(10) = 10 \% 11 = 10$$

$$H(36) = 36 \% 11 = 3$$

$$H(25) = 25 \% 11 = 3$$

$$H(46) = 46 \% 11 = 2$$

For 18, hash address 7 is already occupied, so next free position is 8<sup>th</sup>

43 and 10 has same hash address, so 10 will be inserted at 0<sup>th</sup> position in the table

0	10
1	
2	46
3	36
4	25
5	
6	
7	29
8	18
9	
10	43



# Hashing

## Linear Probing

For Searching the element

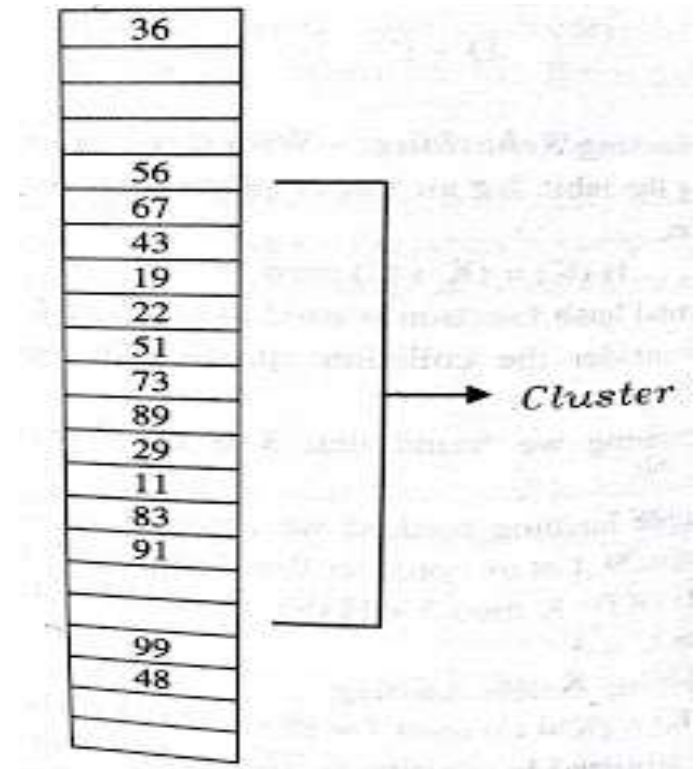
- 1) First we check the hash address position in the table
- 2) If element is not available at that position,
- 3) Sequentially search the element after that hash address position

0	10
1	
2	46
3	36
4	25
5	
6	
7	29
8	18
9	
10	43

# Hashing

## Disadvantage of Linear Probing

- Clustering Problem
- When filled sequence in a hash table becomes longer.
- It means that positions are occupied by elements and we have to search a longer period of time to get an empty cell.
- Searching also becomes slow



# Hashing

## Linear Probing

- 1) If Hash address is  $h$
- 2) In case of collision, linear probing searches the location  $h, h+1, h+2, \dots, (\%SIZE)$

# Hashing

## Linear Probing

- If slot  $\text{hash}(x) \% S$  is full, then we try  $(\text{hash}(x) + 1) \% S$
- If  $(\text{hash}(x) + 1) \% S$  is also full, then we try  $(\text{hash}(x) + 2) \% S$
- If  $(\text{hash}(x) + 2) \% S$  is also full, then we try  $(\text{hash}(x) + 3) \% S$
- $S$ =size of hash table

# Example

- A simple hash function as “key mod 7” and sequence of keys as 50, 700, 76, 85, 92, 73, 101

# Example

- A simple hash function as “key mod 7” and sequence of keys as 50, 700, 76, 85, 92, 73, 101

0	
1	
2	
3	
4	
5	
6	

Initial Empty Table

0	
1	50
2	
3	
4	
5	
6	

Insert 50

0	700
1	50
2	
3	
4	
5	
6	76

Insert 700 and 76

0	700
1	50
2	85
3	
4	
5	
6	76

Insert 85: Collision  
Occurs, insert 85 at  
next free slot.

# Example

- A simple hash function as “key mod 7” and sequence of keys as 50, 700, 76, 85, 92, 73, 101

0	700
1	50
2	85
3	92
4	
5	
6	76

Insert 92, collision occurs as 50 is there at index 1.  
Insert at next free slot

0	700
1	50
2	85
3	92
4	73
5	101
6	76

Insert 73 and 101

# Hashing

## Quadratic Probing

- 1) If Hash address is  $h$
- 2) In case of collision, quadratic probing searches the location  $h+i^2\%SIZE$
- 3) For  $i=1,2,3....$  it searches for location  $h+1, h+4, h+9.....$
- 4) Decreases the clustering problem
- 5) Cannot search all the locations
- 6) If hash table size is prime, it will search atleast half of the locations of the hash table



# Hashing

## Quadratic Probing

- **How Quadratic Probing is done?**

Let  $\text{hash}(x)$  be the slot index computed using the hash function.

- If the slot  $\text{hash}(x) \% S$  is full, then we try  $(\text{hash}(x) + 1*1) \% S$ .
- If  $(\text{hash}(x) + 1*1) \% S$  is also full, then we try  $(\text{hash}(x) + 2*2) \% S$ .
- If  $(\text{hash}(x) + 2*2) \% S$  is also full, then we try  $(\text{hash}(x) + 3*3) \% S$ .
- This process is repeated for all the values of  $i$  until an empty slot is found.
- $S$ =size of the table

# Hashing

## Quadratic Probing

Elements=29,18,43,10,46,54

Table size=11

# Hashing

## Quadratic Probing

Elements=29,18,43,10,46,54

Table size=11

$$H(29)=29\%11=7$$

$$H(18)=18\%11=7, \text{ collision } h+1*1=8$$

0	
1	
2	
3	
4	
5	
6	
7	29
8	18
9	
10	

# Hashing

## Quadratic Probing

Elements=29,18,43,10,46,54

Table size=11

$$H(29)=29\%11=7$$

$$H(18)=18\%11=7, \text{ collision } h+1*1=8$$

$$H(43)=43\%11=10$$

$$H(10)=10\%11=10, \text{ collision } h+1*1=11\%size=11\%11=0$$

0	<b>10</b>
1	
2	
3	
4	
5	
6	
7	29
8	18
9	
10	<b>43</b>

# Hashing

## Quadratic Probing

Elements=29,18,43,10,46,54

Table size=11

$$H(29)=29\%11=7$$

$$H(18)=18\%11=7, \text{ collision } h+1*1=8$$

$$H(43)=43\%11=10$$

$$H(10)=10\%11=10, \text{ collision } h+1*1=11\%size=11\%11=0$$

$$H(46)=46\%11=2$$

$$H(54)=54\%11=10, \text{ collision } h+2*2=h+4=14\%11=3$$

0	10
1	
2	46
3	54
4	
5	
6	
7	29
8	18
9	
10	43

# Hashing

## Quadratic Probing

Elements=29,18,43,10,46,54

Table size=11

0	10
1	
2	46
3	54
4	
5	
6	
7	29
8	18
9	
10	43

# Example

- Let us consider a simple hash function as “**key mod 7**” and sequence of keys as **50, 700, 76, 85, 92, 73, 101**.

# Example

- Let us consider a simple hash function as “**key mod 7**” and sequence of keys as **50, 700, 76, 85, 92, 73, 101**.
- $H(50) = 50 \% 7 = 1$
- $H(700) = 700 \% 7 = 0$
- $H(76) = 76 \% 7 = 6$

## Quadratic Probing Example

0	
1	
2	
3	
4	
5	
6	

Initial Empty Table

0	
1	50
2	
3	
4	
5	
6	

Insert 50

0	700
1	50
2	
3	
4	
5	
6	76

Insert 700  
and 76



# Example

- Let us consider a simple hash function as “**key mod 7**” and sequence of keys as **50, 700, 76, 85, 92, 73, 101**.
- $H(50)=50\%7=1$                        $H(85)=85\%7=1$ , Collision  $(h+1)=2$
- $H(700)=700\%7=0$                $H(92)=92\%7=1$ , Collision again  $(h+4)=5$
- $H(76)=76\%7=6$

0	700
1	50
2	85
3	
4	
5	
6	76

Insert 85:

Collision occurs.

Insert at  $1 + 1*1$  position

0	700
1	50
2	85
3	
4	
5	92
6	76

Insert 92:

Collision occurs at 1.

Collision occurs at  $1 + 1*1$  position

Insert at  $1 + 2*2$  position.

# Example

- Let us consider a simple hash function as “**key mod 7**” and sequence of keys as **50, 700, 76, 85, 92, 73, 101**.
- $H(73)=73\%7=3$
- $H(101)=101\%7=3$ , collision  $(h+1)=4$

0	700
1	50
2	85
3	73
4	101
5	92
6	76

Insert 73 and 101

# Hashing

## Double Hashing

- Hashing 2<sup>nd</sup> time in case of Collision
- **Suppose  $h$  is a hash key, then in case of collision , we again do hashing of this key**
- **$\text{Hash}(h)=h'$**
- **Search the hash key location  $h, h+h', h+2h', h+3h', \dots$**

# Hashing

## Double Hashing

- 2 times calculation of hash function which creates it complex
- **Searching is slower than linear and quadratic probing**
- Very careful in choosing 2<sup>nd</sup> Hash function

# Hashing

## Double Hashing

$$h = \text{key} \% 13$$

$$h' = 11 - (\text{key} \% 11)$$

At the time collision, the next probe will be-

$$= (h + h') \% 13$$

$$= ((\text{key} \% 13) + (11 - (\text{key} \% 11))) \% 13$$

# Hashing

## Double Hashing

- First hash function is typically
  - $\text{hash1}(\text{key}) = \text{key} \% \text{TABLE\_SIZE}$
- A popular second hash function is :
  - $\text{hash2}(\text{key}) = \text{PRIME} - (\text{key} \% \text{PRIME})$
  - where PRIME is a prime smaller than the TABLE\_SIZE.

# Hashing

## Double Hashing

Elements=8, 55, 48, 68

Table size=13

$$h = \text{key} \% 13$$

$$h' = 11 - (\text{key} \% 11)$$

0	
1	
2	
3	55
4	
5	
6	
7	
8	8
9	48
10	
11	
12	68

# Hashing

## Double Hashing

Elements=8, 55, 48, 68

Table size=13

$$H(8) = 8 \% 13 = 8$$

$$H(55) = 55 \% 13 = 3$$

$$H(48) = 48 \% 13 = 9$$

**No collision till now**

0	
1	
2	
3	55
4	
5	
6	
7	
8	8
9	48
10	
11	
12	



# Hashing

## Double Hashing

Elements=8, 55, 48, 68

Table size=13

$$H(8)=8\%13=8$$

$$H(55)=55\%13=3$$

$$H(43)=48\%13=9$$

$$H(10)=68\%13=3, \text{ collision}$$

Next address=

$$=(68\%13)+(11-(68\%11))\%13$$

$$=(3+(11-2))\%13$$

$$=(3+9)\%13$$

$$=12\%13$$

$$=12$$

Position for 68=12

0	
1	
2	
3	55
4	
5	
6	
7	
8	8
9	48
10	
11	
12	68

**Let us take elements 19, 27, 36, 10**  
**Perform Double Hashing using the following functions**

**Lets say,  $\text{Hash1}(\text{key}) = \text{key} \% 13$**

**$\text{Hash2}(\text{key}) = 7 - (\text{key} \% 7)$**

**Lets say, Hash1 (key) = key % 13**

**Hash2 (key) = 7 – (key % 7)**

$$\text{Hash1}(19) = 19 \% 13 = 6$$

$$\text{Hash1}(27) = 27 \% 13 = 1$$

$$\text{Hash1}(36) = 36 \% 13 = 10$$

$$\text{Hash1}(10) = 10 \% 13 = 10$$

$$\text{Hash2}(10) = 7 - (10 \% 7) = 4$$

$$(\text{Hash1}(10) + 1 * \text{Hash2}(10)) \% 13 = 1$$

$$(\text{Hash1}(10) + 2 * \text{Hash2}(10)) \% 13 = 5$$

**Collision**

**h**  
**h'**

**(h+h')mod 13**

**(h+2h')mod 13**

# Hashing

## ReHashing

- Chances of Insertion Failure when table is full
- **Soln=>**
- **Create a new hash table with double size of previous hash table**
- Use the new hash function and Insert all the elements of the previous hash table in the new table

# Hashing

## ReHashing

- Scan the elements of the previous hash table one by one
- Calculate the hash key with new hash function
- Insert them in new hash table

# Hashing

## Rehashing

Elements=7,18,43,10,36,25

Table size=11

$$H(7) = 7 \% 11 = 7$$

$$H(18) = 18 \% 11 = 7, \text{ collision}$$

$$H(43) = 43 \% 11 = 10$$

$$H(36) = 36 \% 11 = 3$$

$$H(10) = 10 \% 11 = 10, \text{ collision}$$

$$H(25) = 25 \% 11 = 3$$

0	10
1	
2	
3	36
4	25
5	
6	
7	7
8	18
9	
10	43

# Hashing

## Rehashing

Elements=7,18,43,10,36,25

Table size=11

Now insert 46

$$H(46) = 46 \% 11 = 2$$

0	10
1	
2	46
3	36
4	25
5	
6	
7	7
8	18
9	
10	43

# Hashing

## Rehashing

- Now to perform Rehashing
- Create a new table
- For Size of New Hash table, We choose the nearest bigger prime number to double the size of the original hash table**

0	10
1	
2	46
3	36
4	25
5	
6	
7	7
8	18
9	
10	43



# Hashing

## Rehashing

- Original Size=11
- Double Size=22
- Nearest Bigger prime no=23
- New Hash function:-
- $H(K)=K \bmod 23$**

0	10
1	
2	46
3	36
4	25
5	
6	
7	7
8	18
9	
10	43

# Hashing

## Rehashing

- New Hash Table-

- $H(K) = K \bmod 23$

$$H(7) = 7 \% 23 = 7$$

$$H(18) = 18 \% 23 = 18$$

$$H(43) = 43 \% 23 = 20$$

$$H(36) = 36 \% 23 = 13$$

$$H(25) = 25 \% 23 = 2$$

Now insert 46

$$H(46) = 46 \% 23 = 23 \% \text{SIZE} = 23 \% 23 = 0$$

0	46
1	
2	25
3	
4	
5	
6	
7	7
8	
9	
10	10
11	
12	
13	36
14	
15	
16	
17	
18	18
19	
20	43
21	
22	

Prof. Shweta Dhawan Chachra

# Hashing

## ReHashing

- Bit more expensive technique but it happens very few times
- Decision of rehashing can be taken on different conditions like
  - **Table is occupied more than half,**
  - **Insertion of new element failure or on any given case**

# Hashing

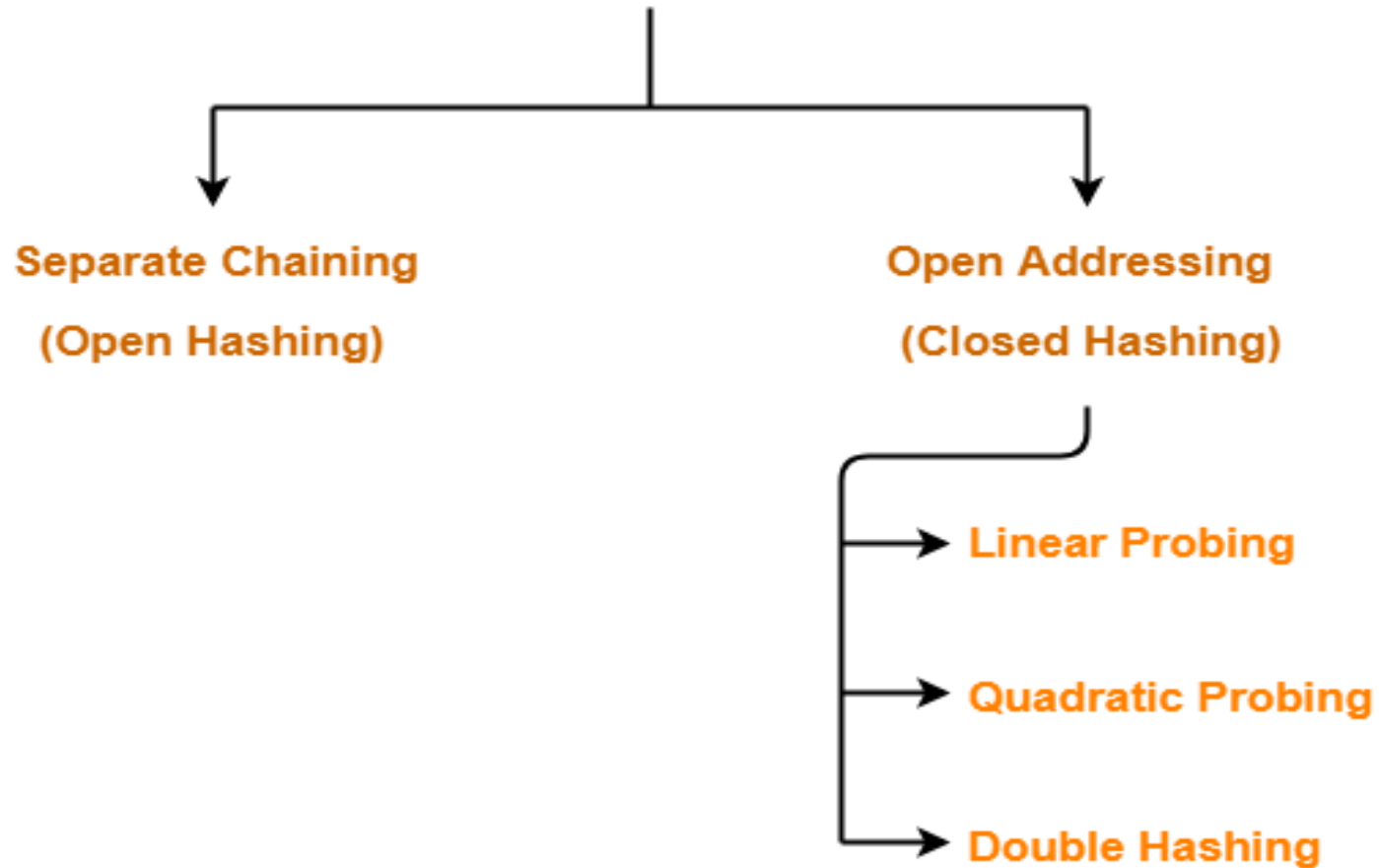
## Collision Resolution (Open Hashing)

### Separate Chaining

- Maintain chain of elements which have same hash address
- Hash table is an array of pointers which point to the linked list
- Maintain Linked list in sorted order and the elements which have same hash address will be in this linked list

# Hashing

## Collision Resolution Techniques



# Hashing

## Collision Resolution (Open Hashing)

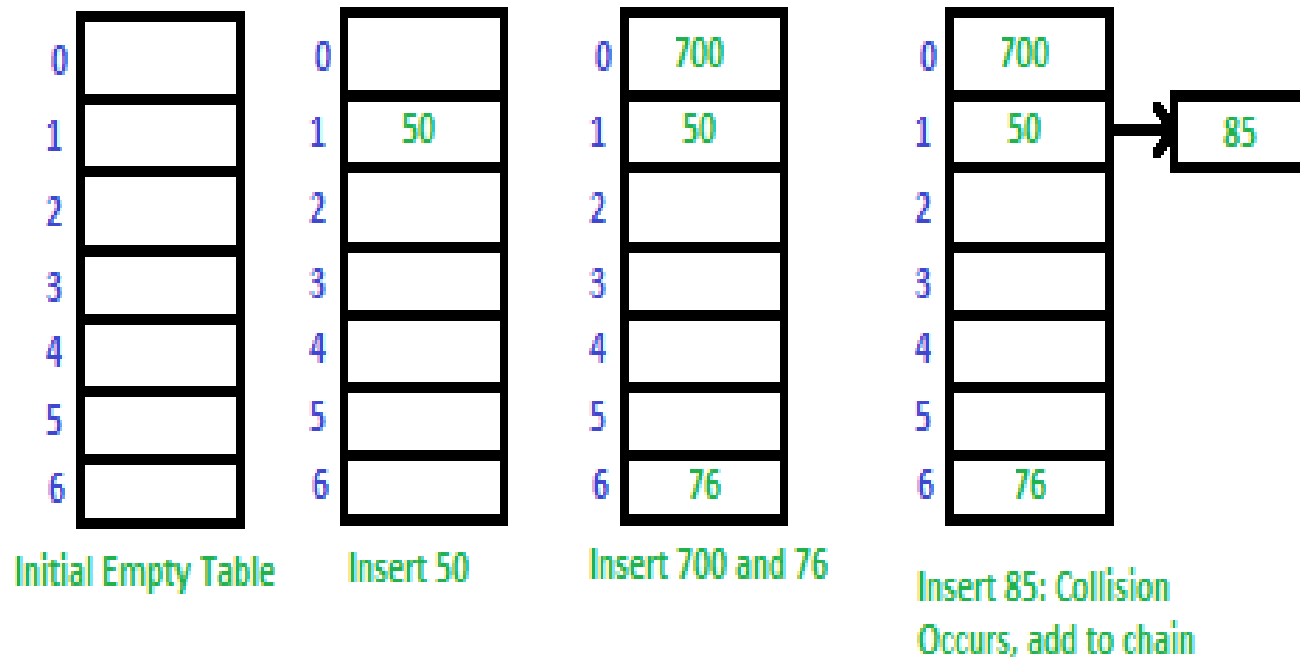
### Separate Chaining

- For inserting one element, we have to get the hash value through the hash function
- Hash value will be mapped in the hash table position then that element will be inserted in the Linked list
- Searching is also same

# Example

## Separate Chaining

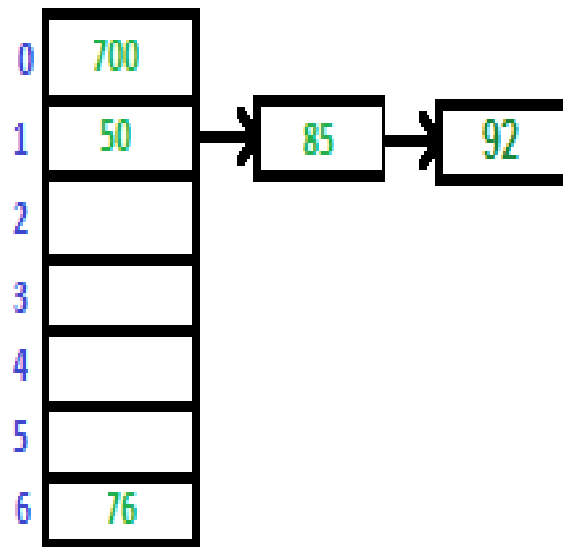
Let us consider a simple hash function as “**key mod 7**” and sequence of keys as 50, 700, 76, 85, 92, 73, 101.



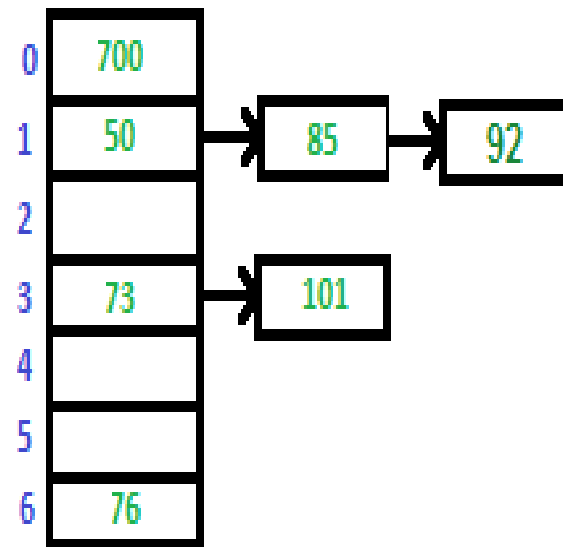
# Example

## Separate Chaining

Let us consider a simple hash function as “**key mod 7**” and sequence of keys as 50, 700, 76, 85, 92, 73, 101.



Insert 92 Collision  
Occurs, add to chain



Insert 73 and 101



# Hashing

## Advantage of Separate Chaining

- 1) Simple to implement.
- 2) Hash table never fills up, we can always add more elements to the chain.
- 3) Less sensitive to the hash function or load factors.
- 4) It is mostly used when it is unknown how many and how frequently keys may be inserted or deleted.

# Hashing

## Disadvantages of Separate Chaining

- 1) **Cache performance of chaining is not good as keys are stored using a linked list. Open addressing provides better cache performance as everything is stored in the same table.**
- 2) Wastage of Space (Some Parts of hash table are never used)
- 3) If the chain becomes long, then search time can become  $O(n)$  in the worst case.
- 4) Uses extra space for links.

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 good as keys are stored using linked list.	Open addressing provides better cache 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