Dictionary

- Dictionary is an ordered collection of pairs of the form<K,V> where K is a key and V is a value associated with the Key. No 2 pairs have the same Key.
- It is container for group of Elements
- Every element of a Dictionary is a Key-Value pair, where Key is unique , Value is associated for each key
- Dictionary is also a Data Structure

Consider a case where we need to store grades of students

For example, the results of a classroom test could be represented as a dictionary with pupil's names as keys and their scores as the values:

'Detra'	17
'Nova'	84
'Charlie'	22
'Henry'	75
'Roxanne'	92
'Elsa'	29

Grades dictionary	
keys	**values**
John	A
Emily	A+
Betty	B
Mike	C
Ashley	A

Key	Value
apple	40
orange	40
pear	50
banana	60

Phone Book

Key	Value
alice	7387
bob	3719
jack	7052
jill	2234

Bank Data

Key	Value
John	3434.34
Tom Smith	123.22
Jane Baker	1378.00
Tod Hall	99.22
Ralph Smith	-19.08

Student Dict

NO	Student object
501	501, 9.5
502	---------
503	----------
504	--------

Operations on Dictionary

- insert
- delete
- find
- size
- isEmpty
- display

Representation of Dictionary:

- Linear List Representation
- Hashing
- Trees:

→Binary Search Trees
→ AVL Trees
→ Red Black Binary Search Trees
→ B Trees

Linear List Representation

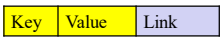
A dictionary can be maintained as order of Key-Value Pairs(where KeY-Value Pairs can be in ascending or descending order based on Keys.

To facilitate this representation, we may implement dictionary as

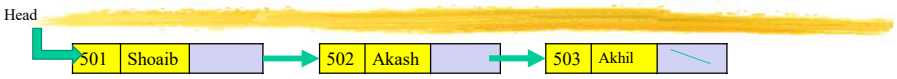
- → **Sorted Array**
- → **Sorted Chain**

Sorted chain:

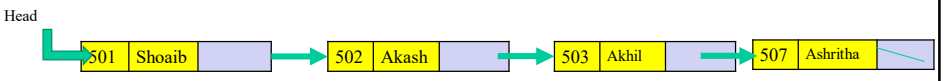
It is a sequence of Nodes that are arranged w.r.t Key(either in ascending or descending)
Each Node holds on Key-Value Pair



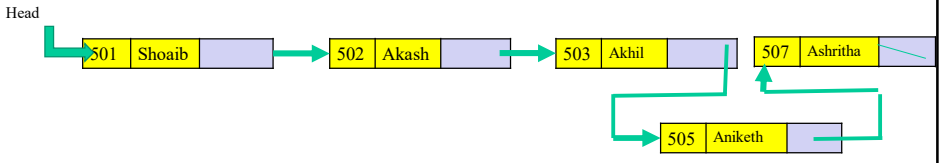
Student Dictionary with Sorted Chain



After adding new record



After adding one morenew record



Dictionary Interface

```
interface DDictionary<K extends
Comparable<K>, V >{
        void insert( Pair<K,V> P);
        void delete(K Key);
        Pair<K,V> find(K Key);
        int size();
        boolean isEmpty();
        void display();
}
```

Implementation of sorted chain

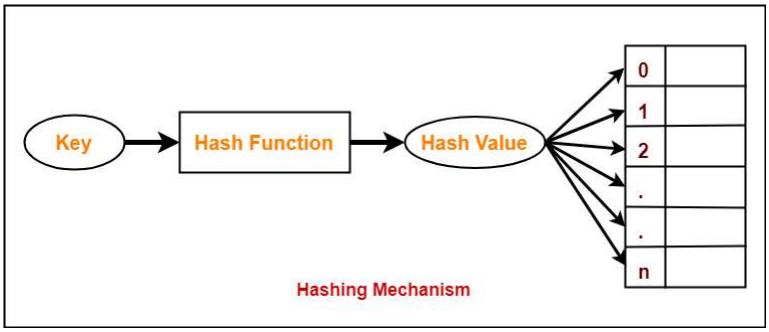
class for Key value pair

```
class Pair<K extends Comparable<K>,V>{
        K Key;
        V Value;
        Pair(){}
        Pair(K Key, V value){
          this.Key=Key;
          this.Value=Value;
        }
        @Override
        public String toString(){
                return " Key  :" + Key+ "Valeue "+Value   ;
        }
}
```

Hashing

Hashing

- Hashing is a process of mapping keys into the hash table by using a hash function.



9

Hash Table

- Hash table is another representation of Dictionary.
- In Hash table, all operations
  - →Insert
  - →delete
  - →search

  are done based on Key
- **Hashing is applied on Key to determine index or position of its associated pair in the Hash Table.**
- Each **position** in **Hash table** is called as **bucket.**
- **Ideal** Hashing performs **insertion**, **deletion** and **search** operations in **O(1) time.** However , the **worst-case** time complexity is **O(n).**
- If **no 2 pairs hash into the same bucket, then it is called as ideal hashing.**
- Ideal hashing **may be possible** with **small size dictionary**. In real-time applications , it is impractical to implement dictionary without collisions

Hash Function

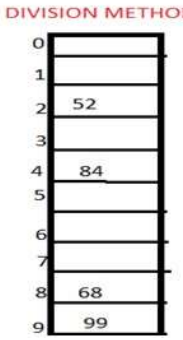
- Hash function maps a key to a location in the table
- Hash function is a mathematical formula that returns hash value.
- Hash value has a fixed length. It is a bucket or position number of a hash table to which a Key is mapped.
- Hash function is denoted by h(x) , where x is a Key.
- The h(x) is called **home bucket.**

11

Types of hash function

Division method

- In this the hash function is dependent upon the remainder of a division. For example:-if the record 52,68,99,84 is to be placed in a hash table and let us take the table size is 10.
- Then:
- h(key)=record% table size.
- 2=52%10
- 8=68%10
- 9=99%10
- 4=84%10



12

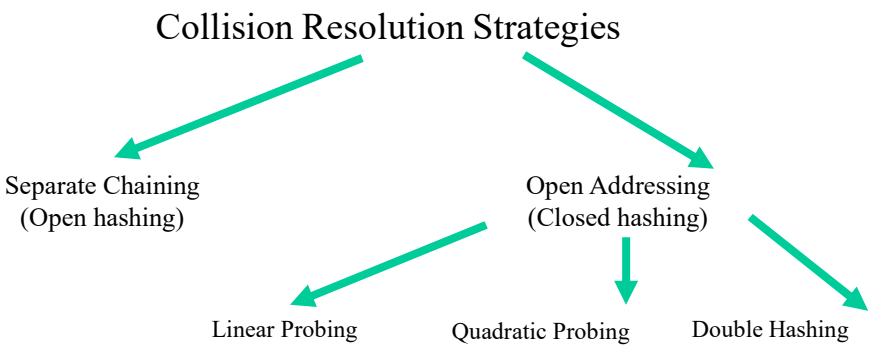
Characteristics of good hashing function

- The hash function must be easy to understand and simple to compute.
- The hash function that gives **good distribution of keys** into hash table is called as **unform hash function**.
- A number of **collisions should be less while placing** the data in the hash table.
- **Efficiency** of hashing depends on **hash function**

13

Collision Resolution Techniques

- Collision occurs when more than one keys (hash functions) map to the same location of hashes.
- There are 2 method for dealing with collisions which are called as collision resolution strategies



14

Open addressing Techniques

- In open addressing techniques ,It probes in the bucket

$$h_0(x), h_1(x), h_2(x), \ldots\ldots\ldots\ldots$$

where $h_i(x)$ =(h(x) + f(i)) % table size

and i is the collision resolution attempt number.

15

Linear Probing:

- ***Linear Probing:*** In linear probing ,it linearly probe for the next slot.

so, In this formula,

$h_i(x)$ =(h(x) + f(i)) % table size

f(i) = i **in linear probing**

16

Example

- Insert the following entries into the Hash table of size 10 using modulo method.

55	A
22	Z
3	C
5	X
75	B
99	A
2	M
9	L
222	K

Initial Hash Table

Example

- Insert the following entries into the Hash table of size 10 using modulo method.

55	A
22	Z
3	C
5	X
75	B
99	A
2	M
9	L
222	K

Step 1:Insertion of (55,A):

55%10 = 5
Place this engtry in 5th Position

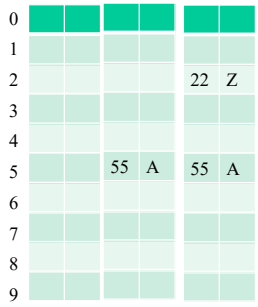
0		
1		
2		
3		
4		
5	55	A
6		
7		
8		
9		

Example

- Insert the following entries into the Hash table of size 10 using modulo method.

55	A
22	Z
3	C
5	X
75	B
99	A
2	M
9	L
222	K

Step 2:Insertion of (22,Z):
22%10 = 2
Place this engtry in 2nd Position

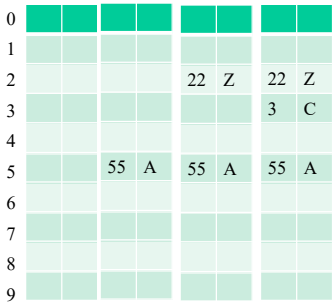


Example

- Insert the following entries into the Hash table of size 10 using modulo method.

55	A
22	Z
3	C
5	X
75	B
99	A
2	M
9	L
222	K

Step 3:Insertion of (3, C):
3%10 = 3
Place this engtry in 3rd Position



Example 2

- Insert the following entries into the Hash table of size 10 using modulo method.

55	A
22	Z
3	C
5	X
75	B
99	A
2	M
9	L
222	K

Step 4:Insertion of (5, X):
5%10 = 5 **collision**

0								
1								
2					22	Z	22	Z
3							3	C
4								
5			55	A	55	A	55	A
6								
7								
8								
9								

5 X
Collision

Example

- Insert the following entries into the Hash table of size 10 using modulo method.

55	A
22	Z
3	C
5	X
75	B
99	A
2	M
9	L
222	K

Step 4:Insertion of (5, X):
5%10 = 5 **collision**
So probe h1(x)= (5 + 1) % 10 =6
Place this engtry in 6th Position

	Initial Hash Table		After insert 55		After insert 22		After insert 3		After insert 5	
0										
1										
2					22	Z	22	Z	22	Z
3							3	C	3	C
4										
5			55	A	55	A	55	A	55	A
6									5	X
7										
8										
9										

Example

- Insert the following entries into the Hash table of size 10 using modulo method.

55	A
22	Z
3	C
5	X
75	B
99	A
2	M
9	L
222	K

Step 5:Insertion of (75, X):
75%10 = 5 **collision**
So probe h1(x)= (5 + 1) % 10 =6 still **collision**
So again probe h2(x)=(5+ 2)%10= 7%10 =7
Place this entry in 7th Position

	Initial Hash Table		After insert 55		After insert 22		After insert 3		After insert 5		After insert 75	
0												
1												
2					22	Z	22	Z	22	Z	22	Z
3							3	C	3	C	3	C
4												
5			55	A	55	A	55	A	55	A	55	A
6									5	X	5	X
7											75	B
8												
9												

Example

- Insert the following entries into the Hash table of size 10 using modulo method.

55	A
22	Z
3	C
5	X
75	B
99	A
2	M
9	L
222	K

Step 5:Insertion of (99, A):
99%10 = 9
Place this engtry in 9 th Position

0														
1														
2					22	Z	22	Z	22	Z	22	Z	22	Z
3							3	C	3	C	3	C	3	C
4														
5			55	A	55	A	55	A	55	A	55	A	55	A
6									5	X	5	X	5	X
7											75	B	75	B
8														
9													99	A

Example

- Insert the following entries into the Hash table of size 10 using modulo method.

55	A
22	Z
3	C
5	X
75	B
99	A
2	M
9	L
222	K

Step 6:Insertion of (2, M):
2%10 = 2 **Collision**
So , probe h1(x) =(2+1)%10 = 3 **Collision**
Probe again h2(x) =(2+ 2)%10 = 4
Place in 4th position

0																			
1																			
2					22	Z	22	Z	22	Z	22	Z	22	Z	22	Z	22	Z	22	Z
3							3	C	3	C	3	C	3	C	3	C	3	C	3	C
4																			2	M
5			55	A	55	A			55	A	55	A	55	A	55	A	55	A	55	A
6							55	A	5	X	5	X	5	X	5	X	5	X	5	X
7											75	B								
8																				
9															99	A	99	A		

Example

- Insert the following entries into the Hash table of size 10 using modulo method.

55	A
22	Z
3	C
5	X
75	B
99	A
2	M
9	L
222	K

Step 7:Insertion of (9, L):
9%10 = 9 **Collision**
So , probe h1(x) =(9+1)%10 = 0
Place in 0th position

0	B																9	L	
1	A																		
2	M				22	Z	22	Z	22	Z	22	Z	22	Z	22	Z	22	Z	22	Z
3	L						3	C	3	C	3	C	3	C	3	C	3	C	3	C
4	K														2	M	2	M		
5			55	A	55	A	55	A	55	A	55	A	55	A	55	A	55	A	55	A
6									5	X	5	X	5	X	5	X	5	X	5	X
7											75	B	75	B	75	B	75	B		
8																				
9													99	A	99	A	99	A		

Example

- Insert the following entries into the Hash table of size 10 using modulo method.

Step 8:Insertion of (222, K):
222%10 = 2 **Collision**
So , probe h1(x) =(2+1)%10 = 3 **Collision**
So , probe again h2(x) =(2+2)%10 = 4 **Collision**
So , probe again h3(x) =(2+3)%10 = **Collision**
So , probe again h4(x) =(2+4)%10 = 6 **Collision**
So , probe again h5(x) =(2+5)%10 = 7 **Collision**
So , probe again h6(x) =(2+6)%10 = 8
Place element at 8 th position

55	A																		
22	Z																		
3	C																		
5	X																		
75	B	0														9	L	9	L
99	A	1																	
2	M	2			22	Z	22	Z	22	Z	22	Z	22	Z	22	Z	22	Z	22	Z
9	L	3					3	C	3	C	3	C	3	C	3	C	3	C	3	C
222	K	4													2	M	2	M	2	M
		5	55	A	55	A	55	A	55	A	55	A	55	A	55	A	55	A	55	A
		6					5	X	5	X	5	X	5	X	5	X	5	X	5	X
		7							75	B	75	B	75	B	75	B	75	B	75	B
		8																	222	K
		9									99	A	99	A	99	A	99	A	99	A

INSERT FOLLOIWNG INTO HASH TABLE OF SIZE 11 .

55,79,80,4,26,36,11,98,3

Step 1:Insertion of (55):
$h_0(55)$= 55%11 = 0
Place this entry in 0th Position

Step 2:Insertion of (79):
$h_0(79)$= 79%11 = 2
Place this entry in 2nd Position

Step 3:Insertion of (80):
$h_0(80)$= 80%11 = 3
Place this entry in 3 Position

Step 4:Insertion of (4):
$h_0(4)$= 4%11 =4
Place this entry in 4th Position

Step 5:Insertion of (26):
$h_0(26)$= 26%11 =4 **collision**
Probe in the next bucket
$h_1(26)$ **=27%11 = 5**
Place this entry in 5th Position

Step 6:Insertion of (36):
$h_0(36)$= 36%11 =3 **collision**
Probe in the next bucket
$h_1(36)$ **=37%11 = 4 collision**
Probe in the next bucket
$h_2(36)$ = 38 %11 =5 **collision**
Probe in the next bucket
$h_3(36)$ = 39 %11 =6
Place this entry in 6th Position

Step 7:Insertion of (11):
$h_0(11)$= 11%11 =0 **collision**
Probe in the next bucket
$h_1(11)$ = 12 %11 =1
Place this entry in 1th Position

Step 8:Insertion of (98):
$h_0(98)$= 98%11 =10
Place this entry in 10th Position

Step 9:Insertion of (3):
$h_0(3)$= 3%11 =3 **collision**
Probe in the next bucket
$h_1(3)$ **=4%11 = 4 collision**
Probe in the next bucket
$h_2(3)$ = 5 %11 =5 **collision**
Probe in the next bucket
$h_3(3)$ = 6%11 =6 **collision**
Probe in the next bucket
$h_4(3)$ = 7%11 =7 **collision**
Place this entry in 7th Position

0	55
1	11
2	77
3	80
4	4
5	26
6	36
7	3
8	
9	
10	98

Exercise problem 2: Given the input

{4371, 1323, 6173, 4199, 4344, 9679, 1989} and hash function

h(x)=x mod 10 show the result for following:

a) open addressing hash table using linear probing

29

- Linear Probing is easy to implement
- In Linear Probing we search sequentially for vacant cells
- The performance is determined by the Load Factor.

Load Factor λ = No. of Key value Pairs / Table Size

Linear Probing performs well as long as load factor <= 0.5

In this strategy ,as more items are inserted in the cluster,cluster grow larger. It is not a problem when the load factor is <=0.5, and still not bad when the load factor is <=0.75. Beyond this, however, the performance degrades seriously as the clusters grow larger and larger.

In linear probing , as long as table is big enough, a free cell can always be found. However, sometimes , a groups of occupied cells are formed : **primary clustering**

- Any hash value inside the cluster adds to the end of that cluster. Hence ,collisions in the cluster get more expensive

Quadratic Probing:

- ***Quadratic Probing:*** Quadratic probing is open addressing strategy. Here the probe function is some quadratic function.

  one possible probe function is

  $f(i) = i^2$

31

1. Given the input

{4371, 1323, 6173, 4199, 4344, 9679, 1989} and hash function

h(x)=x mod 10 show the result for following:

a) open addressing hash table using Quadratic probing

Step 1: insertion(4371)
h(4371)= 4371%10 = 1
Place at 1th position

Step 2: insertion(1323)
h(1323)= 1323%10 = 3
Place at 3rd position

Step 3: insertion(6173)
h(6173)= 6173%10 = 3
collision
h_1(6173)=6174%10=4
4th position

Step 4: insertion(4199)
9th position

Step 5: insertion(4344)
4th position ,collision
5th location is probed and it is empty place in 5th position

Step 6: insertion(9679)
9th position collison
h1(9679) =9680%10=0
so place in 0th position

Step 7: insertion(1989)
9th position collision
h1(1989) =1990%10 =0 collision
h2(1989) = (9 + 4)%10 =3 collision
h3(1989)=(9+ 9)%10=8
place in 8th position

0	9679
1	4371
2	
3	1323
4	6173
5	4344
6	
7	
8	1989
9	4199

Hash the following keys to a hash table of size 8 using quadratic probing:
8,21,56,77,68,5,13

Step 1:Insertion of (8):
$h_0(8)$= 8%8 = 0
Place this entry in 0th Position

Step 2:Insertion of (21):
$h_0(8)$= 21%8 = 5
Place this entry in 5th Position

Step 3:Insertion of (56):
$h_0(56)$= 56%8 = 0 collision
Probe
$h_1(56)$= 57%8 = 1

Step 4:Insertion of (77):
$h_0(77)$= 77%8 = 5 collision
$h_1(77)$= 78%8 = 6

Step 5:Insertion of (68):
$h_0(68)$= 68%8 = 4

Step 6:Insertion of (5):
$h_0(8)$= 5%8 = 5 collision
$h_1(8)$= 6%8 = 6 collision
$h_2(8)$= 9%8 = 1 collision
$h_3(8)$= 13%8 = 5 collision

collisions recurring

0	8
1	56
2	
3	
4	68
5	21
6	77
7	

In Quadrtic Probing

- Load factor of quadratic probing should be <= 0.5
- Quadratic probing eliminates primary clustering problem .
- However quadratic probing form clusters. Clusters of Quadratic probing are called secondary clusters. Secondary clusters size is less compared with primary clusters
- When a collusion occurs, it will probe in the bucket which is some distance away from current bucket.
- Sometimes will never find an empty slot even if table isn't full!
- Luckily, if load factor , $\lambda \leq 0.5$ and table size is prime, guaranteed to find empty slot

Double Hashing

- Double hashing uses 2 hash functions

  (h(x) + i * g(x)) % TABLE_SIZE

  g(x) is the second hash function
- 2nd hash function must never map to 0
- one possible **g(x) =R-x mod R**
- Here, R is prime number, and it is to be less than the table size.
- If $h_0(x)$ is occupied, probe according to

$$f(i) = i \times g(x)$$

- Table size too to be prime number ,otherwise , we may run out of empty buckets permanently.

35

Double Hashing

Example :19,27,36,10 Table size:13
- with g(x)=7 – x mod 7

Step 1: insertion of(19)
h(19)= 19%13=6
Place this entry in 6th Position

Step 2: insertion of(27)
h(27)= 27%13=1
Place this entry in 1th Position

Step 3: insertion of(36)
h(36)= 36%13=10
Place this entry in 10th Position

Step 4: insertion of(10)
h(10)= 10%13=10 collision
g(x)=7-10 %7 =4
h1(x)=(10 +1X4) %13=
14%13=1 collision
h2(x)=(10 +2 X 4) =18%13 =5

0	
1	27
2	
3	
4	
5	10
6	19
7	
8	
9	
10	36
11	
12	

8,21,56,77,68,5,13 ts:13 with R:11

- Insertion of 8
  h(8)=8%13= 8
- Insertion of 21
  h(21)=21%13= 8 collision
  g(x) =11-8%11=3
  h1(21)=(8+ 3) %13 =11
- Insertion of 56
  h(56)=56%13= 4
- Insertion of 77
  h(77)=77%13= 12
- Insertion of 68
  h(8)=68%13= 3
- Insertion of 5
  h(5)=5%13= 8
- Insertion of 13
  h(13)=13%13= 0

0	
1	
2	
3	68
4	56
5	
6	
7	
8	8
9	
10	
11	21
12	77

Rehashing

- If the hash table gets too full , the running time for operations will start taking too long and insertion s might fail for any of the open addressing schemes.
- Solution is to rebuild another hashtable that is twice as big as the current hash table byt select a prime number around the new size
- Then apply rehash. In, Rehash , compute the new hash value for each element and inset it in the new hash table

38

Problem: Rehash the following with linear probing:

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

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

Solution :

new table size is 17

Rehash:

Step 1: $h_0(6)$ =6 %17=6

Step 2: h0(15) =15%17=15

Step 3: h0(23)=23%17=6 collision

probe :

h1(23) = (h0(x) +1)%17 = (6+1)%17

=7%17 =7

Problem: Rehash the following with linear probing:

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

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

Step 4:

h0(24)=24%17=7 collision

probe

h1(24) =(7+1)%17=8%17=8

Step 5: h0(13) =13%17=13

Problem: Rehash the following with linear probing:

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

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

Step 4:

h0(24)=24%17=7 collision

probe

h1(24) =(7+1)%17=8%17=8

Step 5: h0(13) =13%17=13

Rehash with Quadratic Probing

0	**8**	**7**	**17**	**25**		
0	1	2	3	4	5	6

New Table size 17

h(0)=0%17=0

h(8)=8%17=8

h(7)=7%17=7

h(17)=17%17=0 **colllison**

h1(17)= 18%17 =1

h(25)=25%17=8 collision

h1(25)=26%17=9

0	**17**						**7**	**8**	**25**							
0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**

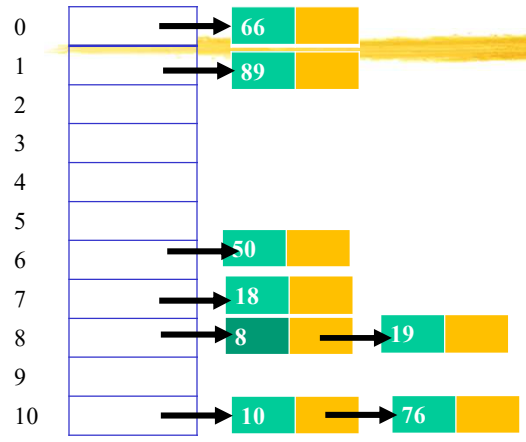
- The running time for rehashing is O(n) where n is the size of the dictionary.
- Rehashing can't be avoided because hash table can't be made arbitrarily large in real time applications
- Rehashing should be done when
  - → the table is half full or
  - → table reaches a certain load factor
  - →when insertion fails

Separate chaining

- In Separate chaining , each cell of hash table point to a linked list of elements that have same hash function value.
- Usually,We keep a sorted linked list of all the elements that hash into the same bucket.
- To perform insertion , first home bucket is determined using hash function and then traverse through the chain associated with the bucket to find an appropriate position for the pair and insert at that position.
- Similar strategy is used for search and deletion

44

Hash the following into separate chaining 8,19,76,50,66,10,89,18



Step 1: Insertion of 8
h(8) = 8%11 =8

Step 2: Insertion of 19
h(19) = 19%11 =8

Step 3: Insertion of 76
h(76) = 76%11 =10

Step 4: Insertion of 50
h(50) = 50%11 =6

Step 5: Insertion of 66
h(66) = 66%11 =0

Step 6: Insertion of 10
= 10%11 =10

Step 7: Insertion of 89
1th Position

Step 8: Insertion of 18
7th Position

Disadvantages:

- Separate chaining has major disadvantage of using linked list.
- So, all the disadvantages associated with linked list are associated with Separate chaining also , like more time for search and more space for storing addresses.
- It uses 2 data structures , Hash table and linked list.
- If the collision rate is high, the search complexity increases as load factor increases