13/08/19. UNIT-2

Hash Tables & Dictionaries.

→ The time complexity for linear search O(n)
→ The time complexity for Binary search O(log n)
→ The time complexity for Hashing O(1) - constant no. of steps.

Hashing :-

Hashing is a technique, used to uniquely identify a specific object from group of similar objects.

Ex :- Library books, identifying with the help of branch & subject.

Hash table :-

It is a data structure in which keys are mapped to array indices (position) by a hash function.

In a Hash table, an element with key k is stored at index h(k).

The process of mapping keys to appropriate positions (locatn) in a hash table is called Hashing.

Ex :-

Index no.	Keys
0	
1	k_1
2	
3	
4	k_2
5	
6	k_3
7	

Hash functions :-

A Hash funct^n is a mathematical formula, when applied to a key, produces an integer, which can be used as an index for the key in the Hash table.

The main Aim of a Hash funct^n is that elements (keys) should be relatively, randomly & uniformly distributed.

It produces a unique set of integers with in some suitable range in order to reduce the no. of collisions.

Note :-
When two or more keys map to the same location then it is said that collision has occured.

A Good Hash function has the following properties

1. Low cost (less complex).
2. Determinism.
3. uniformity.

Different Hash functions.

1. Division Method.

step 1 : find $K \% m = r$. (modular division).

where k is the key

m is the size

Step 2 :- Use r as the index to store the key k.

Q. Given a hash table of size 10. Find the hash value for the keys 12, 14, 16, 17, 18, 19 by using division method.

Sol size (m) = 10.

$K_1 = 12$ $K_2 = 14$, $K_3 = 16$, $K_4 = 17$, $K_5 = 18$, $K_6 = 19$

1. $K_1 \% m = 12 \% 10$.
   $r = 2$.

2. $K_2 \% m = 14 \% 10$.
   $r = 4$.

3. $K_3 \% m = 16 \% 10$.
   $r = 6$.

4. $K_4 \% m = 17 \% 10$.
   $r = 7$

5. $K_5 \% m = 18 \% 10$
   $r = 8$.

6. $K_6 \% m = 19 \% 10$.
   $r = 9$.

Hash table :-

Index.	keys.
0	
1	
2	12
3	
4	14.
5	
6	16
7	17
8	18
9	19.

2. Multiplication Method

Step-1 : choose a const. A such that

$0 < A < 1$.

Step-2 :- Multiply the key k with A

i.e., $k * A$.

Step-3 :- Extract the fractional part of kA.

Step-4 : Multiply the result of step 3 by size of the Hash table (m)

i.e., $h(k) = \lfloor m(kA \bmod 1) \rfloor$

Example :-

$m = 1000$

$k = 12345$

$A = 0.618033$.

$k * A = 12345 \times 0.618033$

$= 7629.617385$

$(k*A)\%1 = 0.617385$.

$[(k*A)\%1] * m = 617.385$

$\lfloor [(k*A)\%1] * m \rfloor = 617$. [integer value].

14/08/2019

Data structures

Hash tables

*. Multiplication Method.

The best choice of A is 0.618033 (suggested by knuth).

3) Mid square Method.

Step 1 :- find the square of the given key i.e., 'k^2'.

step 2 :- Extract the middle r-digits of the result obtained in step 1.

$h(k) = s$

where s is obtained by selecting r-digits from k^2.

Given a hash table of size 10, calculate the hash values for the keys - 11, 13, 14, 16, 17, 20. & insert the keys at resp. locations.

1. $m = 10$

$k_1 = 11$ - $k_1^2 = 121.$ $h(k) = 2$

$k_2 = 13$ $k_2^2 = 169.$ $h(k) = 6.$

$k_3 = 14$ $k_3^2 = 196.$ $h(k) = 9$

$k_4 = 16$ $k_4^2 = 256$. $h(k) = 5$

$k_5 = 17$. $k_5^2 = 289$. $h(k) = 8$

$k_6 = 20$ $k_6^2 = 400$. $h(k) = 0$.

$r = 1$ digit (depends on size).
(middle digit).

index.	key.
0	20
1	
2	11
3	
4	
5	16
6	13.
7	
8	17
9.	14.

if $m = 100$ (we have values 00 - 99)
$r = 2$. (middle 2 no.).

Digital Folding Method.

step 1: Divide the key value into a number of sub parts i.e., divide k in to sub parts $k_1, k_2, \ldots k_n$, where each part has the same no. of digits except the last part, which may have lesser digits than other p step 2: Add the individual subparts i.e., find $k_1 + k_2 + k_3 + \dots k_n$. The hash value is produced by step 2 (ignore last carry if any)

Given a hash table of 100 locations calculate the hash values for the keys, 321, 5678, 34567 by using folding method.

$m = 100$ (0 - 99).

keys	56|78	32|1	34|56|7.
sub parts	56 , 78	32 , 1	34 , 56 , 7
sum.	134.	33	97.

$h(k) = 34$ for $k = 5678$.

(we can ignore 1 from 134., coz max size 99. only).

index	keys
0	
1	
2	
.	
:	
33	321.
34	5678
.	
:	
97	34567
.	

Collisions :-

Collisions occur when two or more keys have same hash value

or

when hash function maps two different keys to the same locatⁿ.

Therefore a method is used to solve the p[roblem] known as Collision Resolution technique.

Collision Resolution by 1. Open Addressing
2. chaining.

Collision Resolution by Open Addressing.

The process of Examining the memory locatⁿ in the hash table is known as probing.

Open Addressing technique can be implemented

1. Linear probing
2. Quadratic probing
3. Double hashing
4. Re-hashing.

1. Linear probing :-

If a value is already stored at the location generated by h(k), then the following hash function is used to resolve the collision

$$\boxed{h(k,i) = (h'(k)+i) \bmod m}$$

where m = size of hash table.

$h'(k) = k \bmod m$ $(k \% m)$.

i is the probe number that varies from $0 - (m-1)$.

Given a hash table of size 10, use linear probing for inserting the keys 72, 27, 36, 63, 92, 81, 101.

$m = 10$

Keys (k) = 72, 27, 36, 63, 92, 81, 101.

$k \% 10$ $(h'(k))$ $= 72 \% 10 = 2.$

$= 27 \% 10 = 7$

$= 36 \% 10 = 6.$

$= 63 \% 10 = 3$

$= 92 \% 10 = 2.$

$= 81 \% 10 = 1$

$= 101 \% 10 = 1$

$i = 0$

(1) $h(72, 0) = (h'(72) + 0) \% 10$ $(h'(72)$

$= 2$ (vacant index locatn)

(2) $h(27, 0) = 7$

(3) $h(36, 0) = 6$

(4) $h(63, 0) = 3$

(5) $h(92, 0) = 2$ (not vacant)

so increment i.

$i = 1$, $h(92, 1) = (h'(92) + 1) \% 10$

$= (2 + 1) \% 10 = 3$ (not

$i = 2$, $h(92, 2) = (h'(92) + 2) \% 10$

$= (2 + 2) \% 10 = 4$ (vac

(6) $h(81, 0) = 1$ (vacant)

7) $h(101, 0) = 1$ (not vacant)

$i = 1$ $(h(101, 1)) = (h'(101) + 1) \% 10$

$= 2 \% 10 = 2$ (not vacant

$i = 2$ $(h(101, 2)) = (h'(101) + 2) \% 10$

$= (1 + 2) \% 10 = 3$ (not vac

$i = 3$ $(h(101, 3)) = (h'(101) + 3) \% 10$

$= (1 + 3) \% 10 = 4$ (not v

$i = 4$ $(h(101, 4)) = (h'(101) + 4) \% 10$

$= (1 + 4) \% 10 = 5$ (vacan

Quadratic probing :-

If a value already stored in a location generated by h(k) then the following hash function is used to resolve the collision.

$$h(k,i) = (h'(k) + c_1 * i + c_2 * i^2) \bmod m$$

where m is size.

$i \rightarrow$ probe number from 0 to m-1,

c_1, c_2 are const. such that $c_1, c_2 \neq 0$

& $h'(k) = k \bmod m$ (k%m)

Searching :-

while searching for a value the array index is recomputed and key of the element stored at that Locatⁿ is compared with the value that has to be searched. If a match is found then the search operatⁿ is succesful otherwise it begins sequential search of the array that continues until

(1) value is found

(2) encounters a vacant locatⁿ

(3) reaches end of the table.

Main draw back of this algorithm is that it results in clustring and thus there is higher risk of more collisions where collisions already taken place

This is also known as primary clustring. So, in order to avoid primary clustring we are going for quadratic probing

→ Searching is similar to linear probing. one of the major draw back of Quadratic probing is that sequence of successive probes may only explore the fraction of the table, on this fractions make quite small.

if this happens

(i) Then we will not be able to find an empty locat^n in the table despite the fact that the table is not full.

This is also known as Secondary clustering Probing.

Double Hashing :-

It uses one hash value and then repeatedly by stops forward, an interval until an empty location is reached. The interval is decided using second independent hash funct^n. Hence it is known as double hashing

$h(k,i) = (h_1(k) + i h_2(k)) \% m.$

$m =$ Size.

$i =$ probe number

$h_1(k) \rightarrow k \% m.$

$h_2(k) = k \% m'$

$(\because m' \text{ is } (m-1) \text{ or } (m-2))$

Example for Quadratic probing

1. consider size of hash table as 10. using Quadratic probing insert keys: 72, 27, 36, 63, 92, 81, 101, in hash table.

take $c_1 = 1$
$c_2 = 3$

0	1	2	3	4	5	6	7	8	9
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1

keys

1. $h(72,0) = (h'(72) + (1\times0 + 3\times0))\ \%\ 10$
$= (2)\ \%\ 10 = 2$ (vacant)

2 ⟶ 72.

2. $h(27,0) = (h'(27) + (1\times0 + 3\times0^2))\ \%\ 10$
$= 7\ \%\ 10 = 7$ (vacant)

7 ⟶ 27.

3. $h(36,0) = (h'(36) + (1\times0 + 3\times0^2))\ \%\ 10$
$= 6\ \%\ 10 = 6$ (vacant).

6 ⟶ 36.

(4) $h(63,0) = (h'(63) + (1\times0 + 3\times0^2))\ \%\ 10$
$= 3\ \%\ 10 = 3$ (vacant).

3 ⟶ 63.

0	1	2	3	4	5	6	7	8	9
-1	-1	72	63	-1	-1	36	27	-1	-1

(5) $h(92,0) = (h'(92) + (1\times0 + 3\times0^2))\ \%\ 10$
$= 2\ \%\ 10 = 2$ (Not vacant).

$h(92, i=1) = (h'(92) + (1\times1 + 3\times1^2)) \% 10$
$= (2 + (1+3)) \% 10 = (6)\%$
$= 6$ (not vacant)

$h(92, i=2) = (h'(92) + (1\times2 + 3\times4)) \% 10$
$= (2 + (2+12)) \% 10 = 16 \% 10$
(not vacant)

$h(92, i=3) = 2$ (not vacant)

$h(92, i=4) = (h'(92) + (1\times4 + 3\times16)) \% 10$
$= (2 + 52) \% 10 = 4$. (vacant)

(6) $h(81, 0) = 1$. (vacant)

(7) $h'(101, 1) = 5$ (vacant).

0	1	2	3	4	5	6	7	8
-1	81	72	63	92	101	36	27	-1

Example for Double hashing.

Q Hash table of $m=10$, use double hashing, in the keys 71, 27, 36, 63, 92, 81, 101
take $m' = (m-1)$

Sol

$m' = (m-1) = 9$.
$m = 10$.

$h(k,i) = (h_1(k) + i.h_2(k)) \% m$.

$h_1 = k \% m$; $h_2 = k \% (m-1)$.

0	1	2	3	4	5	6	7	8
-1	-1	-1	-1	-1	-1	-1	-1	-1

$h(72,0) = (h_1(72) + 0 * h_2(72)) \% 10$

$(h_1 = 72 \% 10 = 2)$

$= (2) \% 10 = 2$ (vacant)

(2) $\longrightarrow$ 72.

$h(27,0) = (h_1(27) + 0 * h_2(27)) \% 10$

$= 7 \% 10 = 7$ (vacant)

$h(36,0) = (h_1(36) + 0 * h_2(36)) \% 10$

$= 6 \% 10 = 6$ (vacant)

$h(63,0) = (h_1(63) + 0 * h_2(63)) \% 10$

$= 3 \% 10 = 3$ (vacant)

0	1	2	3	4	5	6	7	8	9.
-1	-1	72	63	-1	-1	36	27	-1	-1

$h(92,0) = (h_1(92) + 0 * h_2(92)) \% 10.$

$= 2 \% 10 = 2$ (not vacant)

$h(92,1) = (h_1(92) + 1 \times h_2(92)) \% 10$

$= (2 + 1 * 2) \% 10 = 4$ (vacant).

$(h_2 = 92 \% 9 = 2.)$

$h(81,0) = (h_1(81) + 0 * h_2(81)) \% 10$

$= 1 \% 10 = 1$ (vacant)

$h(101,0) = (h_1(101) + 0 * h_2(101)) \% 10$

$= (1) \% 10 = 1$ (not vacant)

$h(101,1) = (h_1(101) + 1 \times h_2(101)) \% 10$

$= (1 + 3) \% 10 = 4$ (not vacant)

$h(101,2) = (h_1(101) + 2 \times h_2(101)) \% 10.$

$= (1 + 2 * 2) \% 10 = 5$ (vacant)

20/8/19.

* Rehashing :-

→ Load factor = $\frac{n}{m}$.

where n = no. of entries.
m = size of hash table

→ default Load factor value = 0.75

Q Given the hash table of size 5, insert the following keys - 20, 31, 42, 53, 37, 46, 65. in to the hash table by using rehashing. Take the Load factor has 0.75 & $h(k) = k \% m$.

Sol.

$h(20) = 20 \% 5 = 0$.

$h(31) = 1$.

$h(42) = 2$

$h(53) = 3$

m=5

0	20
1	31
2	42
3	53
4	

calulate load factor = $\frac{\text{no. of entries}}{\text{size}} = \frac{4}{5} = 0.8$

∴ Load factor (0.8) > default load factor

Therefore we go for rehashing.

Now $m' = m \times 2 = 5 \times 2 = 10$

$h(20) = 20 \% 10 = 0$

$h(31) = 1$

$h(42) = 2$

$h(53) = 3$

$h(37) = 7$

$h(46) = 6$

$h(65) = 5$

m'=10

0	20
1	31
2	42
3	53
4	
5	65
6	46
7	37
8	
9	

Once again calculate load factor.

$$= \frac{n}{m} = \frac{7}{10} = 0.7 \le 0.75.$$

* When load factor (α) rises more than it's predefined value (0.75), the complexity increases and thereby degrading the performance of insertion and search operation.

In such cases, a better option is to create a new hash table with the size double of the original hash table.

All the entries in the original hash table will then have to be moved to the new hash table.

Collision Resolution by Chaining. (chained hash table)

In chaining each location in the hash table stores a pointer to a linked list that contains all the key values that were hashed to that location.

i.e., location 1 in the hash table points to the head of the linked list of all key values that hashed to location 1.

However, if no key values hashed to 1 then location 1 in the hash table contains NULL

Q. Insert the keys 7, 24, 18, 52, 36, 54, 11, 23 in a chained hash table of 9 memory locations use. $h(k) = k \% m$.

0	NULL
1	NULL
2	NULL
3	NULL
4	NULL
5	NULL
6	NULL
7	NULL
8	NULL

– before inserting keys.

$m = (0-8) = 9.$

$\therefore$ $h(k) = k \% m$

$h(7) = 7 \% 9 = 7$

$h(24) = 6$

$h(18) = 0$

$h(52) = 7$

$h(36) = 0$

$h(54) = 0$

$h(11) = 2.$

$h(23) = 5.$

0		→ 18 → 36 → 54
1	NULL	
2		→ 11 / NULL
3	NULL	
4	NULL	
5		→ 23 / NULL
6		→ 24 / NULL
7		→ 7 → 52 / NULL
8	NULL	

* The Main advantage of using a chained hash table
- it remains effective, even when the no. of keys to be stored is much higher than size of hash table
- however, with the (↑)se in the no. of keys to be stored, the performance of the chained hash table degrade gradually.
- This technique is absolutely free from clustering problem.

Assignment:

Q Given a hash table of size 10, insert keys

21, 32, 44, 29, 68, 71, 94, 109, 14

1. by using linear probing
2. by using Quadratic probing
3. Double hashing
4. rehashing
5. chained hash table

$h(k) = k \% m$

C: Quadratic probing $c_1 = 2, \ c_2 = 3$

double hashing $m = 10, \ m' = 9$

21/08/2019

Extendible Hashing

Q. Insert the keys 8, 25, 12, 7, 6, 33, 19, 21, 34, 50, 49 by using Extendible Hashing. use h(k) = k % 16

g = 2 & l = 2

Bucket size is '2'

Key	h(key) $\boxed{k \% 16}$	Binary number
8	8	1000
25	9	1001
12	12	1100
7	7	0111
6	6	0110
33	1	0001
19	3	0011
21	5	0101
34	2	0010
50	2	0010
49	1	0001.

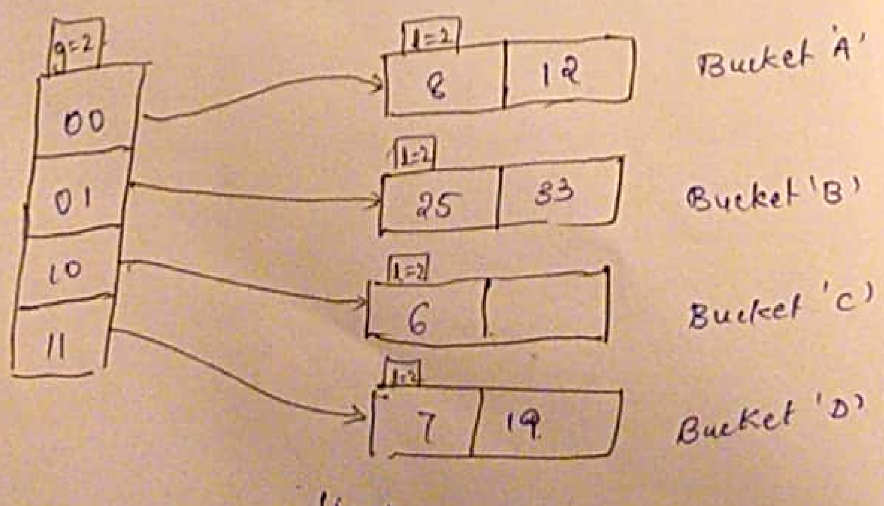
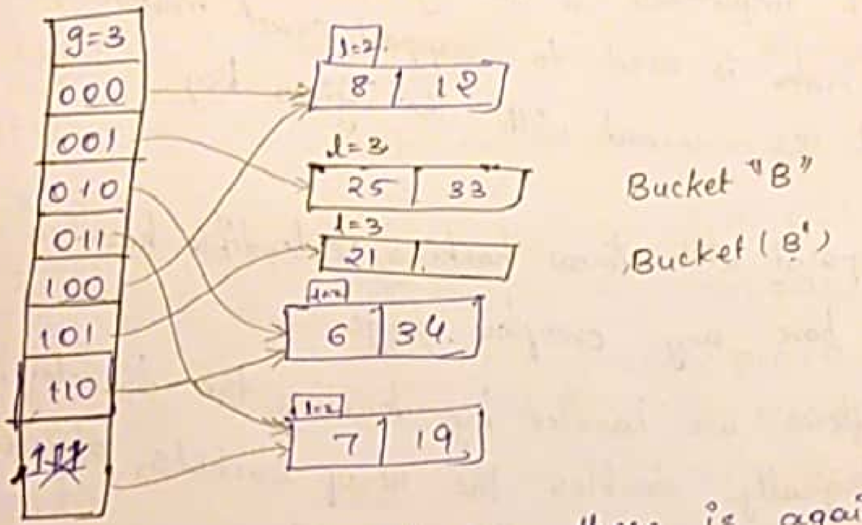


fig-1.

fig 1 indicates insertion of keys 8, 12, 25, 33, 6, 7, 19 whereas incase of key 21 (0101) points to Bucket 'B'.

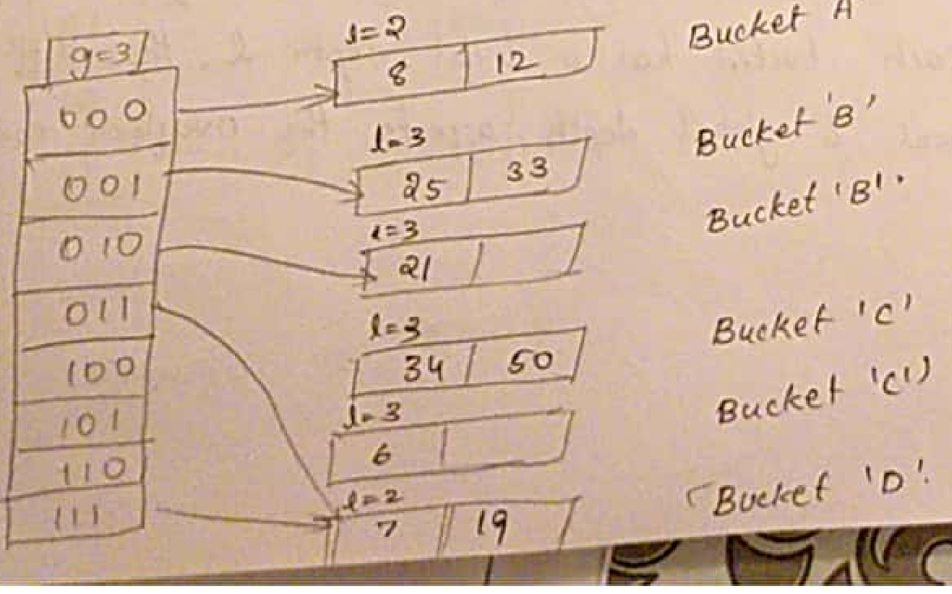
This creates overflow condition (as Bucket 'B' is full) so go for rehashing with g=3



In order to insert 50, there is again a overflow condition as bucket 'c' is full.

To overcome the situation, split the Bucket "c" in to "C" & "C'" as 010 & 110 both are pointing to Bucket "c", & check last 3 digits of binary code.

Now, after spliting, 010 points to Bucket "c"
110 points to Bucket "c'"



In order to insert 49, we again have an overflow condition

---

* It is a dynamically updatable disk base, index str... which implements a hashing a directory.
The index is used to support exact match queries i.e., find the record with the given key
*
Compared with linear hashing, extendible hashing doesn't have any overflow page.
* Overflows are handled by doubling the directory, which logically doubles the no. of buckets, physically only overflown bucket is split.
* Multiple keys may be hashed to the same bucket
* It uses a directory to access it's buckets, the directory is enough to kept an array with 2^d entries.
where d = global depth of the directory. To decide where a key k is stored, extendile hashing uses last d bits of the hash funct^n h(k)
* each bucket has a local depth l, the difference b/w local & global depth, affects the overflown condition

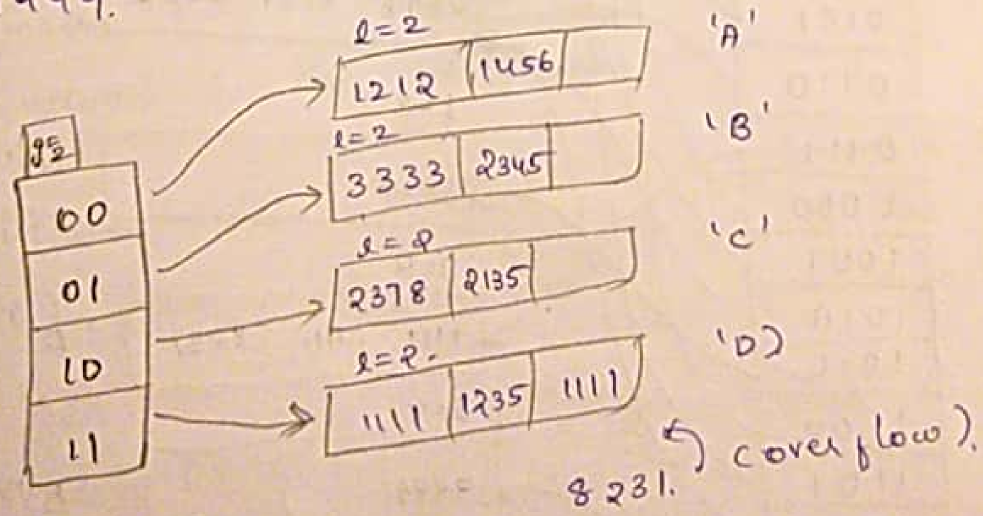
Q. Insert the keys 1111, 3333, 1235, 2378, 1456, 2134, 1212, 2345, 1111, ~~828~~ 8231, 2222, 9999 using Extendible hashing, use $h(k) = k \% 64$

$g = 2$ & $l = 2$.

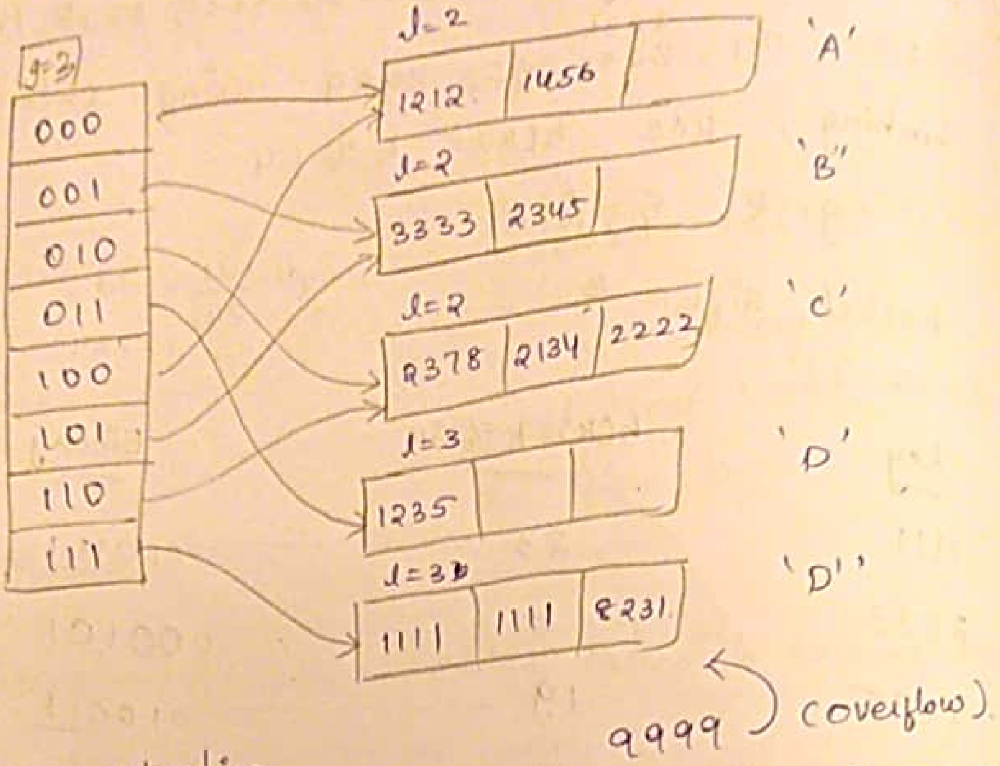
bucket size = 3

($\frac{1111}{64} = 17.$ $17 \times 64 - 1111$ [in calci]

key	$h(k) = k \% 64$	Binary
1111	23	010111
3333	5	000101
1235	19	010011
2378	10	001010
1212	60	111100
1456	~~60~~ 48	110000
2134	22	10110
2345	41	101001
1111	23	010111
8231	39	100111
2222	46	101110
9999	15	001111

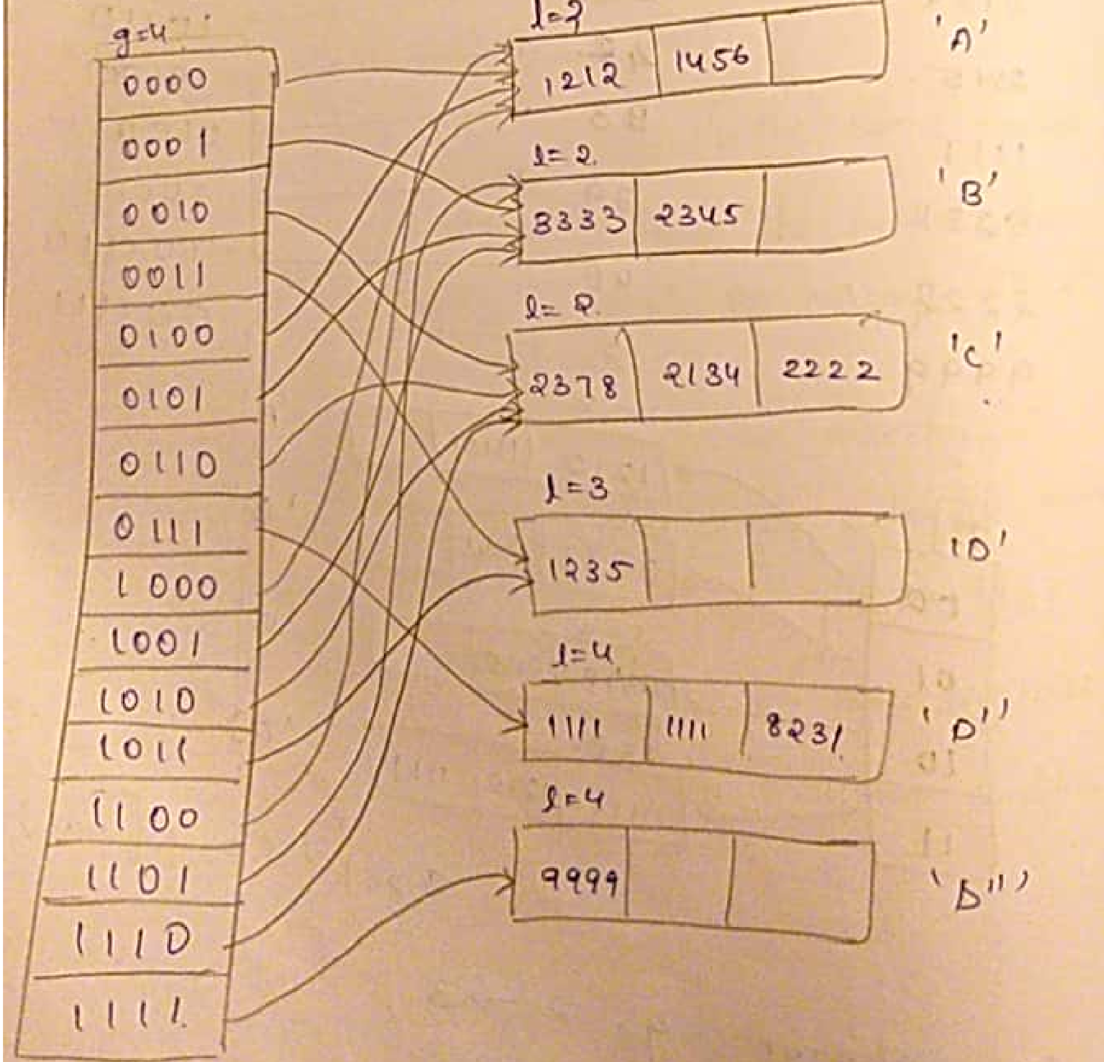


so, change g: 2 → 3,



go for rehashing.

Change g : 3 ⟶ 4, overflow condition is occuring at D', so split the D' in to two, after rehasing.



Assignment :-

Insert the following keys

- 3, 5, 7, 10, 15, 31, 44, 64, 92, 55, 63, 17, 13

use $h(k) = k$ $g = 2$, $l = 2$ bucket size = 4,

22/8/19.

Dictionaries :

A dictionary is an ordered list of key element pairs, where keys are used to locate elements in the list

Ex :- consider a data structure that stores Bank account details.

It can be viewed as a dictionary where account numbers serve as keys for identific-ation of details of account.

possible operations :-

* insert (x, D) :- insertion of an element (x)(key) & (value) in the ~~dectorary~~ dictionary D.

* Delete (x, D) - deletion of an elemen x (key & value) from the dictionary D with the help of keys.

* Search (x, D) - searching of a value x in the dictionary D with a key.

* Member (x, D) – it returns true if x belongs to D else returns false.

* sizeof(D) : It returns the total no. of elements in the dictionary D.

Example :-

Operation	Dictionary	Output
insert(5,A)	{(5,A) }.	(5,A)
[(5,A) is x.]		
insert (7,B)	{ (5,A) (7,B)}	{(5,A) (7,B)}
insert (1,C)	{(1,C) (5,A) (7,B)}	{(1,C) (5,A) (7,B)}
search (7) (key)	{ (1,C) (5,A) (7,B)}	B (returns value)
Member (6)	{(1,C) (5,A) (7,B)}	false (there is no key, which is =6)
Delete (1)	{(1,C) (5,A) (7,B)}	{ (5,A) (7,B)}
sizeof( )	{ (5,A) ,(7,B) }	2.

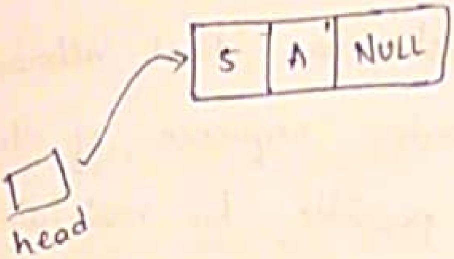
(Note :- In the above example , the no. of dictionaries are assumed to be 1)

Representation of Dictionary using Linear List
(sorted chain)

A Linear List is the collection of (key, value) pairs,

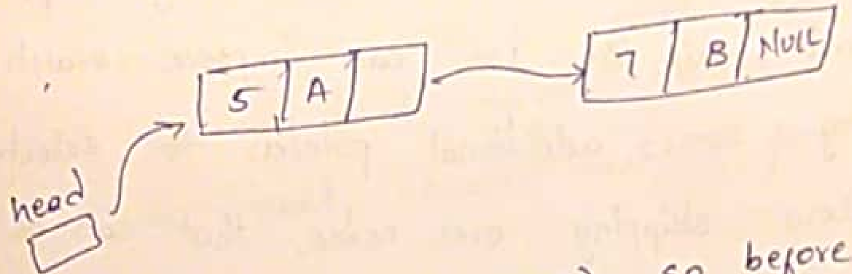
Sorted chain :- the content of the Dictionary is always in sorted form.

(a) insert (5, A)

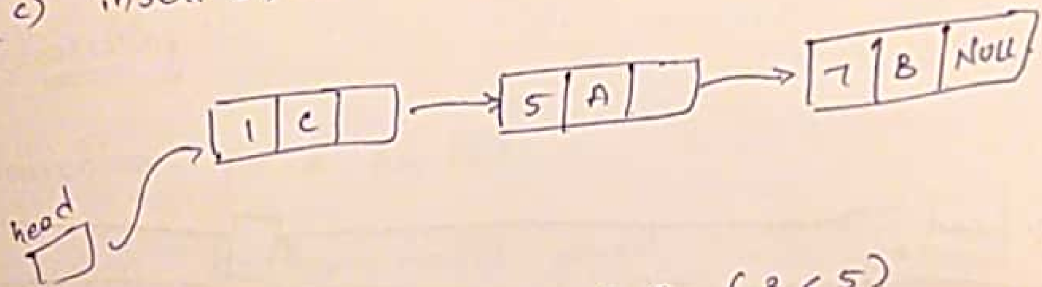


(b) insert (7, B)

(5 < 7), so after 5.

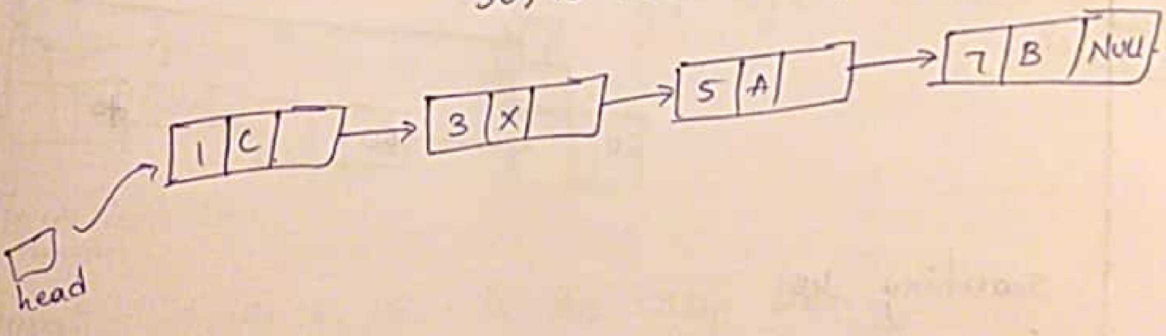


(c) insert (1, C). (1 < 5), so before 5.



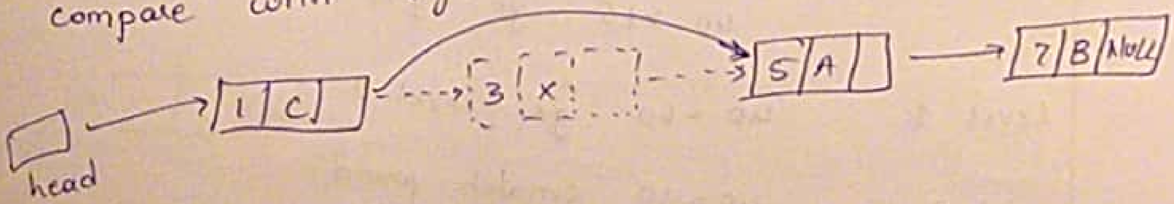
d. insert (3, X) (1 < 3) & (3 < 5)

so, is inserted after 1 & before 5.

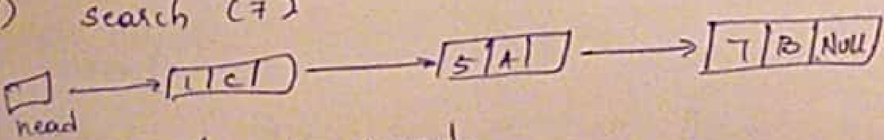


(e) delete (3)

compare with keys. (1 ≠ 3), (3 = 3).



(f) search (7)



element found.

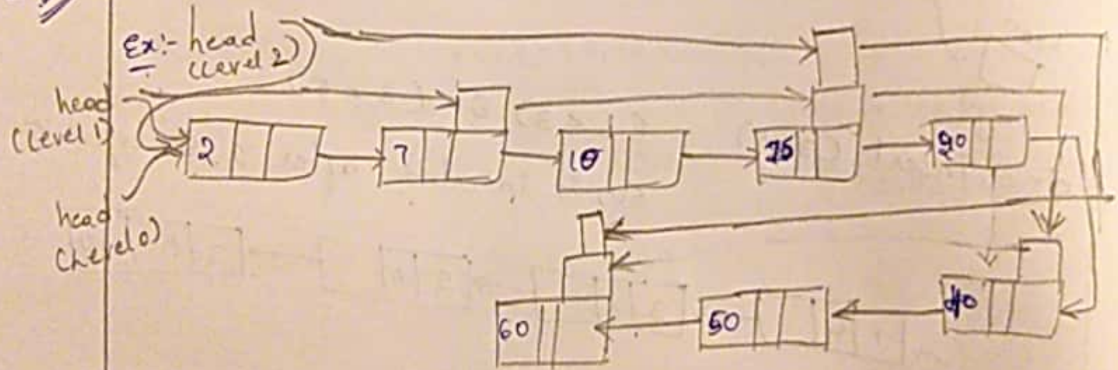
Skip List

A skip list is a data structure that allows fast search with in an order sequence of elements. fast search is made possible by maintaining a Linked hierarchy of sub sequence.

→ Linked List data structures are relatively simple to implement skip List. We can improve search performance by adding some additional pointers to selected nodes to allow skipping over nodes, that can be safely ignored.

26/8/19

Ex:-



searching 40.

(Level 2) compare with 15.
40 > 15 go right

Level 1. 40 < 60 go level 0

Level 0. 40 = 40 (match found)

Notes :-

1. Ideal skip List has the following properties.

a. all the nodes are in Level 0, half of the nodes are in Level 1, 1/4 of the nodes are in Level 2 and so on.

b. Nodes in Level 0 point to the next node, nodes in Level 1 point to the next & the second next node, nodes in Level 2 points to next, second next & fourth next node

Searching :-

Searching for a key (k)

1.) if k= key , match found.

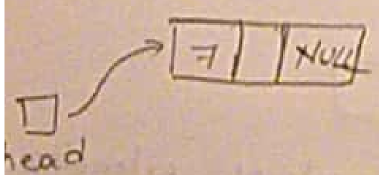
2) if k < key (next key) go down a Level

3) if k > next key go right in the same level

Insertions :-

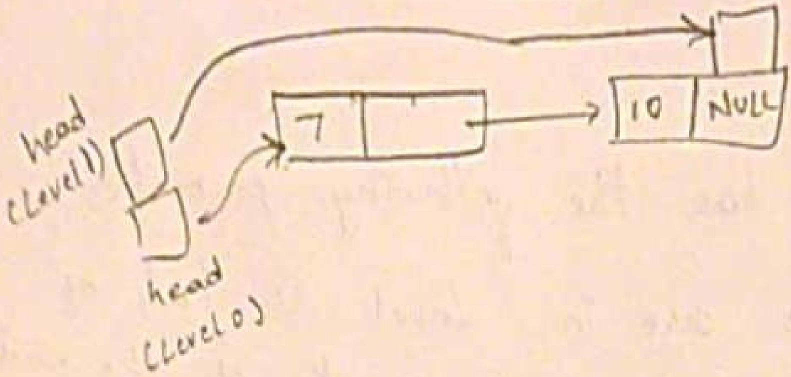
Insert 7, 10, 2, 15 in the skip List.

1. Insert 7.



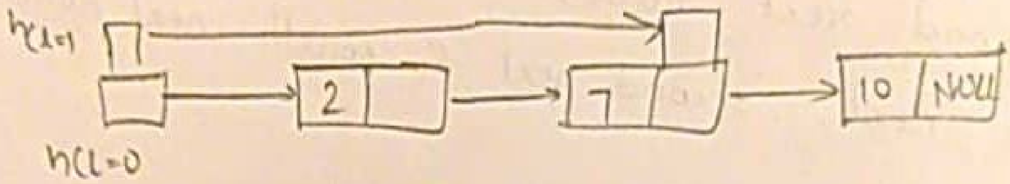
2. Insert 10.

check, 7>10 (false) , so insert after 7.



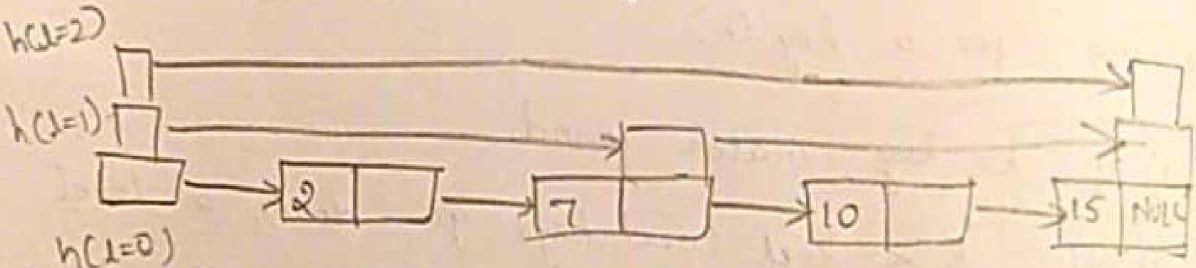
3. insert 2.

2 < 7, insert before 7.



(4) insert 15,

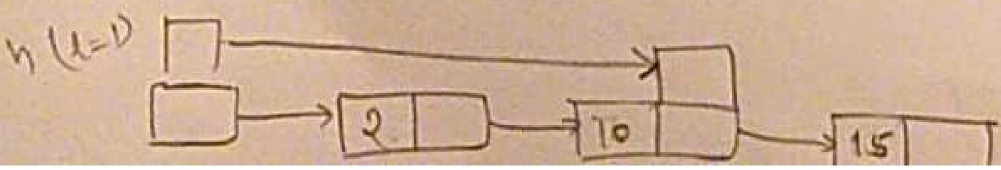
15 > 10, insert after 10.



(5) Deletion.

a. Delete 7

1. find the element 7., then change connect^n



Applications of Hash tables / Dictionaries :-

1. It is used for database indexing.
2. It is used to implement compiler symbol tables
3. In many data base systems file & directory hashing is used to obtain high performance.
4. password verification syst^m.
5. CD databases, file signatures.
6. Game boards