16/01/24 TUE.

UNIT : NO : 4 : (ONE) :-

HASHING

# Hashing.

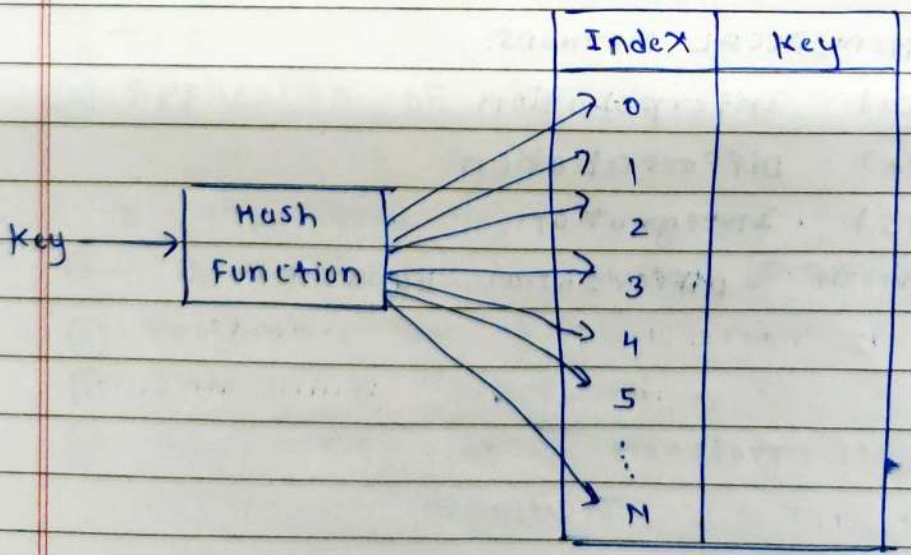
→ Hashing is the process of indexing and retrieving element in a data structure to provide faster way of finding element using the hash key.

→ The main objective of hashing is to get time complexity O(1)

→ With hashing we get O(1) search time on average and O(n) search time on worst case.

# HASH TABLE :-

- Hash Table is a datastructure used for storing and retrieving data quickly.
- All data is inserted into hash table based on the hash key-value.
- It is used to Map the Data with the index in the hash table.



PAGE NO.
DATE :

- **Buckets:** A Bucket in a hash File is a unit of storage (typically a disk block) that can hold one or more records. The hash table consists of 'b' buckets and each bucket consists of s slots. (Usually s = 1)
- **Overflow:** When Hashtable becomes ~~Full~~ full and new record needs to be inserted then it is called Overflow. An Overflow occurs when we hash a new identifier into a Full bucket.
- **Perfect hash function:** It is a function that maps distinct key elements into hash table with no collision
- **Load Factor or Load Density of Hash Table:-**

→ $LD = n/m$

n = no. of elements stored in table.
m = size of table.

# **Hash Functions:-**

→ ∘ A Good Hash Function should:
- Minimize collisions
- Be Easy and quick to compute.
- Distribute key values evenly in the hash table.
- Use all the information provided in the key.

PAGE NO.
DATE :

# Division Method:-

- Idea:
  - Compute hash value from key using the % operator
  - Map a key k into one of the m slots by taking the remainder of k divided by m.

$$h(k) = k \bmod m$$

Example: $k = 1276$, $n = 10$

$h(1276)$ ~~m~~ $= 1276 \bmod 10$

$= 6$

Advantage:

Fast, requires only one operation.

DisAdvantage:

Certain values of m are not a good choice. For Eg power of 2. Take size i.e. a power of 2 like 32 and 1024 should be avoided, for it leads to more collision.

# Multiplication Method:-

- Idea:
  - Multiply key k by a constant A, where $0 < A < 1$
  - Extract the fractional part of kA and multiply the fractional part by m
  - Take the floor of the result.

$$h(k) = \lfloor m (kA \bmod 1) \rfloor$$

fractional part of $kA = kA - \lfloor kA \rfloor$

Example:- $k = 123$, $m = 100$, $A = 0.618033$

$h(123) = 100 (123 * 0.618033 \bmod 1)$

DATE :

$= 100\ (76.018059 \bmod 1)$

$= 100\ (0.018059) = 1$

$h(123) = 100(0.018059) = 1$

* Digit Extraction Method.

- Selected Digits are extracted from the key and used as address

  Address = selected digits from key.

Eg:- If six digit Employee no, is 379245 then then first digit as the index so 379 is the Key Address

\# FOLDING:-

→ 1.) Fold shifting Method
2.) Fold Bounding Method.

(1.) Fold shifting :-

→ Here, Actual values of each part of the key are added

Eg:- 12345678

∴ 12|34|56|78

= 12 + 34 + 56 + 78

= 180

↳ neglect.

= 80 ... (Ans)

↳ key.

Ex:-(2) 25936715

⟹ 25 | 93 | 67 | 15 → Divide into 2

∴ 25 + 93 + 67 + 15

= 200

↳ neglect

= 00 → hash key value.

⟹ 2593 | 6715 → Divide into 4

= 2593 + 6715

= 9308

↳ hash key value.

(2.) Fold Bounding Method:-

→ Here reversed values of divided part are added.

Eg:- 12345678

⇒ 12|34|56|78 → Divide into 2 group.

Now, By reversing and adding.

21 + 43 + 65 + 87

= 216

→ 216

↳ Neglect the first (msb) digit.

= 16 → (hash value.)

Neglect those many digits which are out of bound Neglect the first (msb bit) digit which is '2' and '16' will be considered. The splitting is done in 2 part thus the MSB will be omitted.

17/01/24

# MID SQUARE METHOD:-

→ Idea:

- The key is squared and the middle part of the result taken as the hash table.
- To map the key 3121 into a hash table of size 1000, we square it $3121^2 = 9740641$ and extract 406 as the hash value.

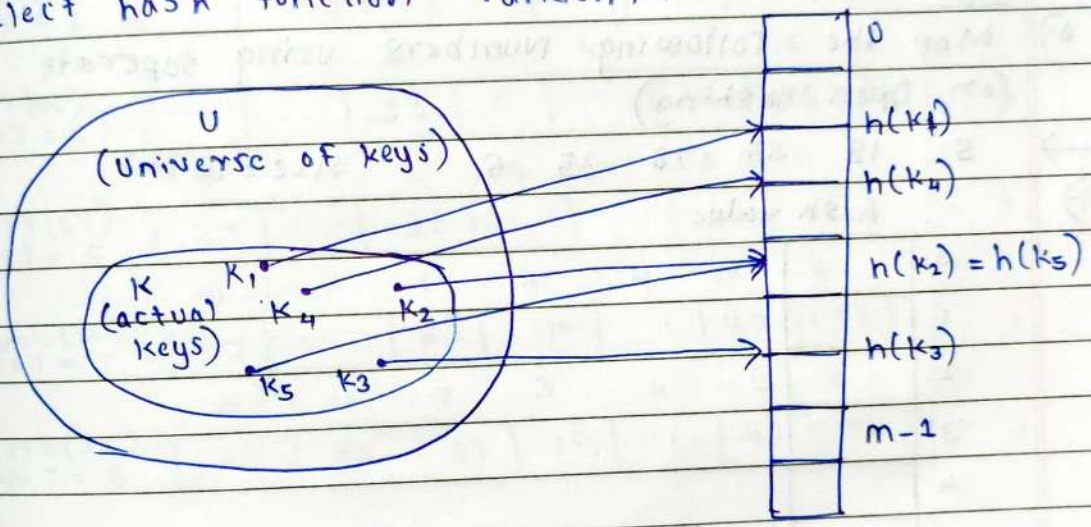
- Advantage: Works well if the keys do not contain a lot of leading or trailing zeroes.
- Dis-Advantage:
  - Selection of Middle Part.
  - Non-Integer Keys have to be pre-processed to corresponding Integer.

Ex:- To Map 3121 into a hash table of 1000, we square it $3121^2 = 9740641$ and Extract 406 as the hash value.

DATE :

# UNIVERSAL HASHING:-

- For any hash function if the table size m is much smaller than universe size U, then for any hash function h, there is some large subset of U that has the some hash value.
- So we need set of hash functions and select hash function random.




# COLLISION HANDLING TECHNIQUES:-

- Seperate chaining.
- Open Addressing.
  - Linear Probing.
  - Quadratic Probing.
  - Double Probing.

28/01/24

(1) Seperate Chaining:-

→
- It is also called as 'Open - Hashing'
- In this method, a seperate list of all elements map to the same value (hash value) is maintained.
- If the memory space is very less, then this method is not feasible.
- Additional space is required to store addresses of head arrays.

PAGE
DATE :

PROS

- Collision resolution is Simple.
- No problem of load factor can hold more number of elements.
- Table size need not be prime number.

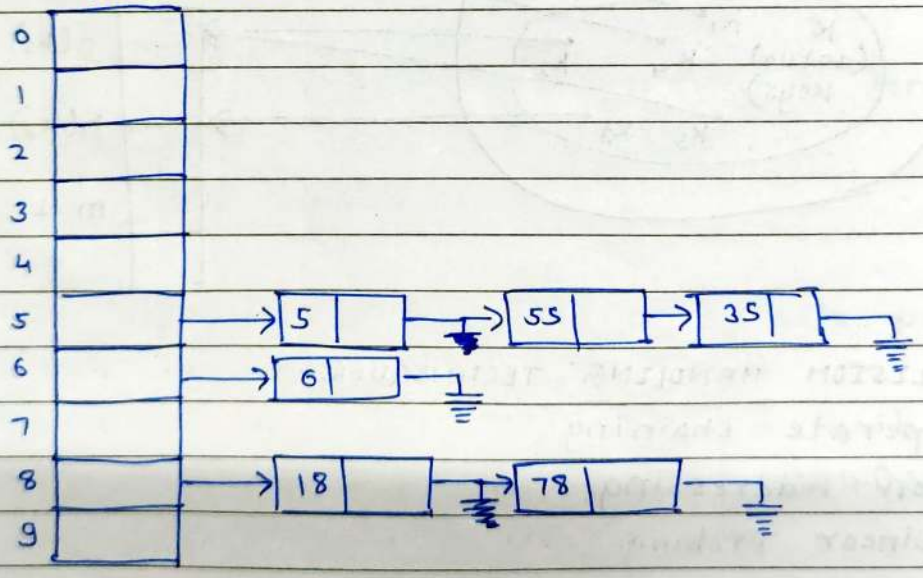
Eg:- (SUM)

Q) Map the following Numbers using seperate chaining. (or Open hashing)

→ 5 18 55 78 35 6 size = 10 (n)

A)

hash value




7
10 ) 78
−70
8

# OPEN ADDRESSING (closed Hashing):-

→ (1) Linear Probing.

- Table remains a simple array of size m.
- On Insert (x),

  First Compute h(x) = x mod m,

  if collision occur

For Eg:- $m = \{76, 93, 40, 47, 10, 55\}$

$h(x) = x \bmod m$

size (m) = 7.

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | Probes |
|---|---|---|---|---|---|---|---|---|
| Insert(76) 76 % 7 = 6 | | | | | | | 76 | 1 |
| Insert(93) 93 % 7 = 2 | | | 93 | | | | 76 | 1 |
| Insert(40) 40 % 7 = 5 | | | 93 | | | 40 | 76 | 1 |
| Insert(47) 47 % 7 = 5 | 47 | | 93 | | | 40 | 76 | 3 |
| Insert(10) 10 % 7 = 3 | 47 | | 93 | 10 | | 40 | 76 | 1 |
| Insert(55) 55 % 7 = 6 | 47 | 55 | 93 | 10 | | 40 | 76 | 3 |

→ Types of Linear Probing.

① Linear Probing with chaining (without Replacement)

- Excessive collision when occurs it becomes very difficult to maintain indexes of same hash key
- Extra field is added to maintain chain

Example:- Let M = 10, H(X) = X MOD M

KEYS = {0, 1, 4, 71, 64, 89, 11, 33}

| | Key | chain |
|---|---|---|
| 0 | 0 | -1 |
| 1 | 1 | 2 |
| 2 | 71 | 3 |
| 3 | 11 | 1 |
| 4 | 4 | 5 |
| 5 | 64 | -1 |
| 6 | 33 | -1 |
| 7 | | -1 |
| 8 | | -1 |
| 9 | 89 | -1 |

Keys = {12, 1, 4, 3, 7, 8, 10, 5, 14, 6, 28}

$h = x \bmod m$

Now, $12 \% 10 = 2$ $\quad 7 \% 10 = 7$ $\quad 5 \% 10 = 5$

$1 \% 10 = 1$ $\quad 8 \% 10 = 8$ $\quad 14 \% 10 = 4$

$4 \% 10 = 4$ $\quad 10 \% 10 = 0$

$3 \% 10 = 3$ $\quad 2 \% 10 = 2$

| | Key | chain |
|---|---|---|
| ⇒ 0 | 10 | -1 |
| 1 | 1 | -1 |
| 2 | 12 | 5 |
| 3 | 3 | -1 |
| 4 | 4 | 9 |
| 5 | 2 | 6 |
| 6 | 5 | -1 |
| 7 | 7 | -1 |
| 8 | 8 | -1 |
| 9 | 14 | -1 |

1.) $12 \rightarrow 2$

2.) $4 \rightarrow 14$

3.) $2 \rightarrow 5$

6 and 28 input cannot be added as table size is Full

② linear probing with chaining (with replacement)

→ • Problem of misplaced starting location of the chain is handled

• Extra Field is added to maintain chain

Eg:- Let M = 10 (size)

H(x) = X MOD M

KEYS = {0, 1, 4, 71, 64, 89, 11, 33}

if add 22 : 1 → 11 → 77.

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| → Key | 0 | 1 | 71 | 11 | 4 | 64 | | | | 89 |
| Index | -1 | 2 | 3 | | 5 | | | | | |

PAGE NO.
DATE

Q) Insert following numbers into hash table where table size is 10 using Linear probing. with or without replacement.

→ Keys :- 11 33 20 88 79 ~~89~~ 98 44 68 66 22

Without Replacement

| | Key | Index |
|---|---|---|
| 0 | 20 | -1 |
| 1 | 11 | -1 |
| 2 | 98 | 5 |
| 3 | 33 | -1 |
| 4 | 44 | -1 |
| 5 | 68 | ~~7~~ -1 |
| 6 | 66 | -1 |
| 7 | 22 | -1 |
| 8 | 88 | 2 |
| 9 | ~~79~~ | -1 |

88 → 98 → 68
↓
88 → 68 → 98

With Replacement

| | Key | chain | | |
|---|---|---|---|---|
| 0 | 20 | -1 | 20 | -1 |
| 1 | 11 | -1 | 11 | -1 |
| 2 | 98 | 5 | 22 | -1 |
| 3 | 33 | -1 | 33 | -1 |
| 4 | 44 | -1 | 44 | -1 |
| 5 | 68 | -1 | 68 | 7 |
| 6 | 66 | -1 | 66 | -1 |
| 7 | 22 | -1 | 98 | -1 |
| 8 | 88 | 2 | 88 | 5 |
| 9 | 79 | -1 | 79 | -1 |

# 2) Quadratic Probing:-

→ One way to reduce primary clustering is to use Quadratic probing to solve collision.

- We start from the original hash location i.
- If the local

- In Quadratic probing, the location j, (j+1), (j+4), (j+9), .... are examined to find the first empty cell where the key is to be inserted.
-

PAGE NO.
DATE :

Q.] Map the following keys using Quadratic probing where the table size is 11. The keys are:-

{ 20, 30, 2, 13, 25, 24, 10, 9 }

→ $h(x) = x \bmod m$

A.]

| i | keys |
|---|---|
| 0 | 9 |
| 1 | |
| 2 | #2 ✓ |
| 3 | 13 ✓ |
| 4 | 25 ✓ |
| 5 | |
| 6 | 24 |
| 7 | 7 |
| 8 | 30 ✓ |
| 9 | 20 ✓ |
| 10 | 10 |

① $h(20) = 20 \bmod 11 = 9$

$= 9$

② $h(30) = 30 \bmod 11 = 8$

③ $h(2) = 2 \bmod 11 = 2$

④ $h(13) = 13 \bmod 11 = 2$ ...(collision)

⑤ $h(25) = 25 \bmod 11 = 3$ ...(collision)

⑥ $h(24) = 24 \bmod 11 = 2$

⑦ $h(10) = 10 \bmod 11 = 10$

⑧ $h(9) = 9 \bmod 11 = 9$ ...(collision)

→ 10 → 0

1
11 ) 20
11

0
20 ) 11

2
11 ) 30
22
8

0
11 ) 2
-0
2

2
11 ) 25
-22
3

Q) Map the Following keys using quadratic probing.

→ Table - size : 7 , $h(x) = x \bmod m$

Keys : { 76, 40, 48, 50, 55 }

⇒

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | Probe |
|---|---|---|---|---|---|---|---|---|
| Insert(76) 76%7 = 6 | | | | | | | 76 | 1 |
| Insert(40) 40%7 = 5 | | | | | | 40 | 76 | 1 |
| Insert(48) 48%7 = 6 | 48 | | | | | 40 | 76 | 2 |
| Insert(50) 50%7 = 5 | 48 | 50 | 5 | | | 40 | 76 | 3 |
| Insert(55) 55%7 = 6 | 48 | | 5 | 55 | | 40 | 76 | 3 |

# DOUBLE HASHING :-

- It reduces the clustering in a better way.
- Use primary hash function $h_1(k)$ to determine the first slot.
- Use a second hash function $h_2(k)$ to determine the increment for the probe sequence.

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

where $i = 0, 1, 2, \ldots$

- Intial probe : $h_1(k)$
- Second probe is offset by $h_2(k) \bmod m$, so on...
- Advantage : avoids clustering.

Q) Insert the keys using Double hashing technique

Table-size : 11 , $h(x) = x \bmod m$ , $h_2(x) = 7 - (x \bmod 7)$

Keys : { 58, 14, 9, 45 }

1 2 5

→

| | |
|---|---|
| 0 | |
| 1 | |
| 2 | |
| 3 | 58 ✓ |
| 4 | |
| 5 | |
| 6 | 91 ✓ |
| 7 | |
| 8 | |
| 9 | 25 ✓ |
| 10 | 14 ✓ |

$h_1(58) = 58 \bmod 11$
$= 3$

$h_1(14) = 14 \bmod 11$
$= 3$

$\therefore h_2(14) = 7 - (14 \bmod 7)$
$= 7 - 0 = 7$

$\therefore h(k,i) = (3 + 1 \times 7) \bmod 11$
$= 11 \bmod 11 = 10$

$h_1(91) = 91 \bmod 11$
$= 3$

$h_2(91) = 7 - (91 \bmod 7)$
$= 7 - 0 = 7$

$h(k,i) = (3 + 2(7)) \bmod 11$
$= 17 \bmod 11 = 6$

$h_1(25) = 25 \bmod 11 = 3$

$h_2(25) = 7 - (25 \bmod 7)$
$= 7 - 4 = 3$

$h_2(25) = (3 + 2 \times 3) \% 11$
$= 9 \bmod 11$

$\boxed{h_2(25) = 9}$

Q) Keys = { 76, 40, 48, 58, 55 }

m, (size) : 7

$h_1(x) = x \bmod m$

$h_2(x) = 5 - (x \bmod 5)$

→ SOLUTION:-

$h_1(76) = 76 \bmod 7$

$\boxed{h_1(76) = 6}$

$h_1(40) = 40 \bmod 7$

$h_1(40) = 5$

| | |
|---|---|
| 0 | |
| 1 | 48 |
| 2 | |
| 3 | 58 |
| 4 | 55 |
| 5 | 40 |
| 6 | 6 |

PAGE NO.
DATE :

$h_1 \text{mod}(48) = 48 \bmod 7$

$= 6$

$h_2(48) = 5 - (48 \bmod 5)$

$= 5 - 3 = 2$

$h(48, i) = (6 + 1 \times 2) \bmod 7$

$= 8 \bmod 7 = 1$

$h_1(5) = 5 \bmod 7 = 5$ ... (Collision)

$h_2(5) = 5 - (5 \bmod 5)$

$= 5 - 0 = 5$

$h(5, i) = (5 + 1 \times 5) \bmod 7 = 10 \bmod 7 = 3$

$h_1(55) = 55 \bmod 7 = 6$ ... (Collision)

$h_2(55) = 5 - (55 \bmod 5)$

$= 5 - 0 = 5$

$h_2(55) = 5$

$h(55, i) = (6 + 1 \times 5) \bmod 7$

$= 11 \bmod 7 = 4$

# Overflow :-

→ When the load factor (L.F.) is greater than equal to one it is known as 'Overflow.'

# Re-hashing :-

→ • Do hashing again.
• Rehashing : 1.) Overflow
2.) Load Factor
• It means increasing the size of the table and again hashing the keys.
• Drawback : Unnecessary overhead.

# Applications of Hashing.

1.) Database Systems.
2.) Books Records.
3.) Data Dictionaries.
4.) Symbol Tables.

# Applications of Hash Table :-

1.) Network processing Algorithms.
2.) File Systems.
3.) Password Verification.
4.) Pattern Matching.

# Rehashing :-

→ • The most common rehashing technique is to construct a new table of approximately double size of the original hash table.
• Rehashing is a costly operation and it happens frequently when the hash table is small and there are lot of insertions.
• The time required for rehashing is O(N) since

N elements need to be rehashed from the original hash table into the new one.

When to Rehash:

1. Rehash when table is half full
2. Rehash as soon as insertion fails.
3. Rehash beyond a certain load factor 1.

Advantages of Rehashing:

1. No need to consider the table size while inserting data.
2. Hash tables cannot be made arbitrarily large to start with in complex programs.
3. Rehashing can be used for other data structures as well.

# Extendable Hashing:-

- Dynamic hashing methods.
- Bulky data : Disk accesses increased.
- Extendable hashing reduces disk accesses while retrieving the data. It handle large amount of data.

⇒ ▫ Global depth : It is associated with the directories. They denote the number of bits which are used by the hash function to categorize the keys.

| Global Depth = Number of bits in directory ID |
|---|

▫ Local Depth : Local Depth is associated with the buckets Local depth in accordance with the global depth is used to decide the action to be performed in case an overFlow occurs. Local Depth is always less than or equal to the Global Depth.

▫ Bucket Splitting: When the Number of elements in a bucket exceeds a particular size, then the bucket is split into two parts.

PAGE NO.
DATE :

Directory Expansion: Directory expansion takes place when a bucket overflows.

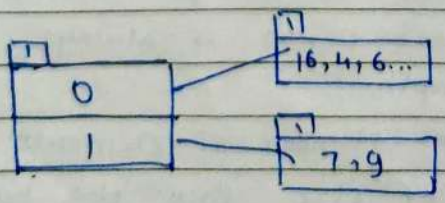
STEPS:-

(1.) Analyze the data elements (type)
(Convert the data type to binary format.)
(2.) Check the Global depth of the dictionary.
(3.) Identify the dictionary.
(4.) Navigation.
(5.) Insertion and Overflow check.
(6.) Tackling Overflow condition during data insertion.
(7.) If required, do rehashing of split bucket elements.

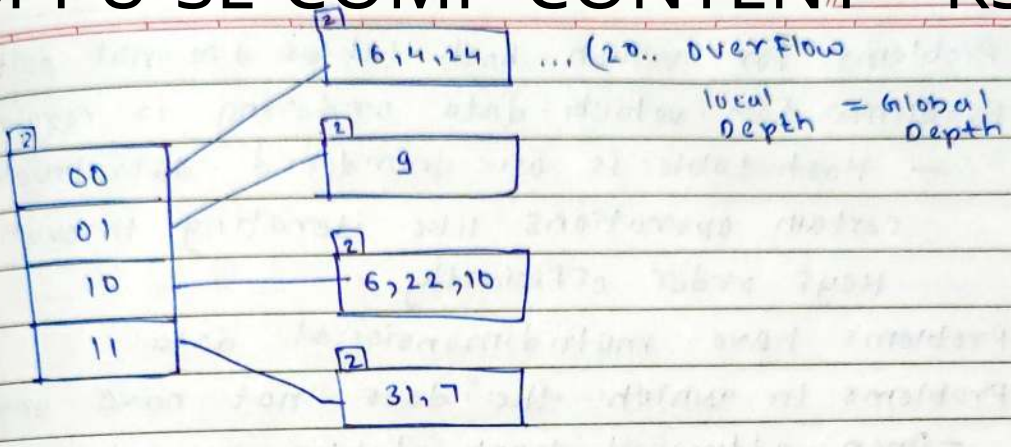
Eg:- Elements : 16, 4, 6, 22, 24, 10, 31, 7, 9, 20, 26.
Bucket size : 3 (Apssumc)

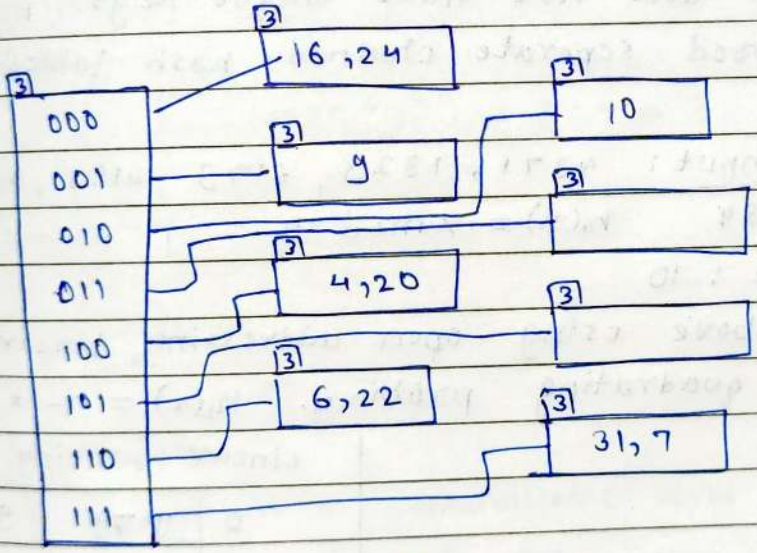
- Hash Function : Suppose the Global depth is X. Then the hash function returns X LSB's

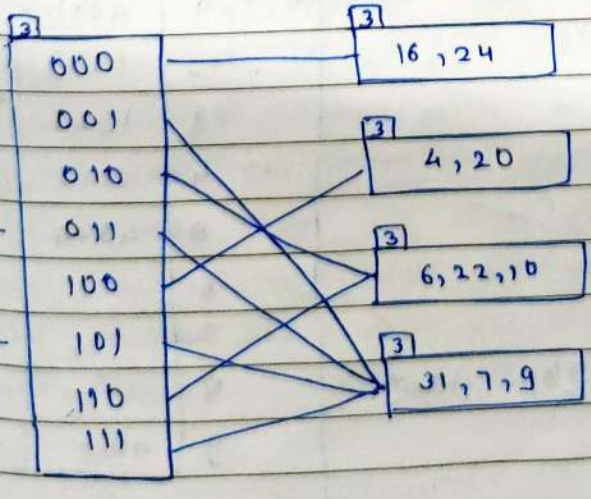
| | Decimal | Binary | Decimal | Binary |
|---|---|---|---|---|
| STEP : I : | 16 | 10000 | 10 | 01010 |
| | 4 | 00100 | 31 | 11111 |
| | 6 | 00110 | 7 | 00111 |
| | 22 | 10110 | 9 | 01001 |
| | 24 | 11000 | 20 | 10100. |




(22,.. Overflow condition.
Here, local depth = Global Depth



# Problems for which hash tables are not suitable.

- Problems for which data ordering is required.
  - Hash table is an unordered datastructure, certain operations like iterating through the keys order efficiently.
- Problems have multidimensional data.
- Problems in which the does not have unique keys.
  - Open-addressed hash tables cannot be used if the data does not have unique keys. An alternative is used seperate chained hash tables.

Given the input: 4371, 1323, 6173, 4199, 4344, 9679, 1989 $h_1(x) = x \bmod 10$

Table size : 10

Solve the above using open addressing, Linear probing or quadrating probing. $h_2(x) = 7 - x \bmod 7$

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

Linear probing.

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

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|
| | 4371 | | 1323 | 6173 | | | | | 4199 |

Double - hashing.

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|
| | 4371 | | 1323 | 6173 | | | 4344 | | 4199 |

① 4371 % 10 → 1

② 1323 % 10 → 3

③ 6173 % 10 → 3 → 9

7 - (6173 mod 7) = 7 - 6 = 1

④ 4199 % 10 → 9

⑤ 4344 % 10 → 4

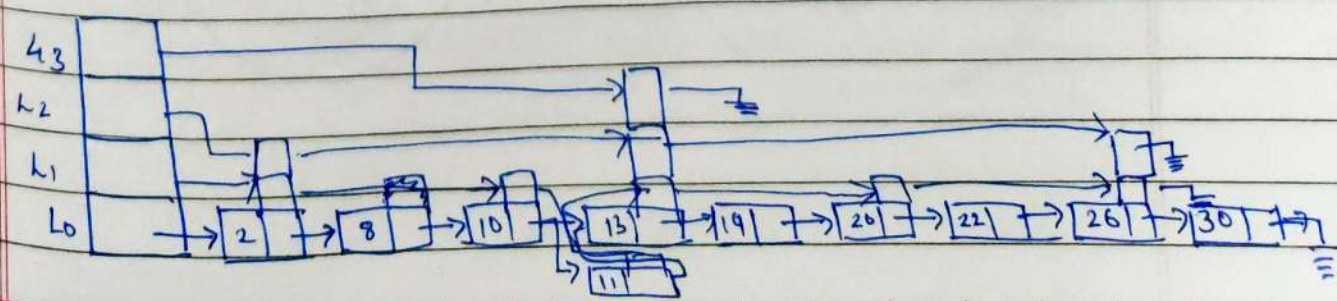
7 - (4344 % 7) = 7 - 4 = 3

⑥

# SKIP-LIST:-

→ A skip list is a probabilistic data structure and an extended version of the linked list.

- The skip-list is used to store a sorted list of elements or data with a linked list and very useful for concurrently accessing element.
- In single step, it skip elements of the entire list, hence referred as skip list
- It allows the user to search, remove and insert the element very quickly.



PAGE NO.
DATE:

# The Operation performed on skiplist are:-

→ (1.) Insertion Operation: It is used to add a new node

(2.) Deletion Operation: It is used to delete a node in a specific solution.

(3.) Search Operation: The search operation is used to search a particular node in a skiplist

(4.) Average case time complexity of all above operations O(logn) →

o Application of skiplist.

→ It is used in distributed applications. In distributed systems, the nodes of skiplist represents the computer systems and pointers represent network connection.