Hashing

$h(x) = x \% 5$

↓

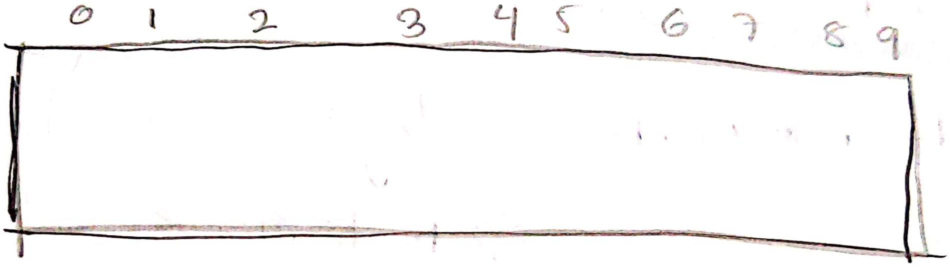
hash function

Phonepe :- → Yes bank

< 6-8 ~~Years~~ hours

→ ICICI

S ⇒ 10



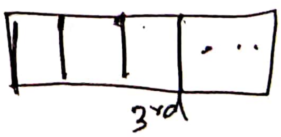
203, 500, 707, 24, 84, 60

mobile numbers ⇒ 6

no is 203

⇒ Size of table is 10

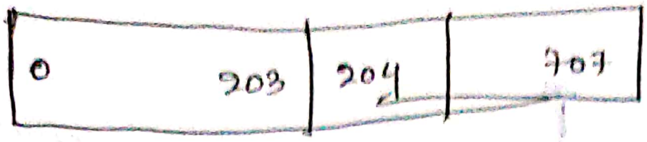
⇒ 203 % 10 → 3 at Index



Size of table is 10

= 500 % 10 ⇒

Collision ⇒ when two different numbers are having same hash value

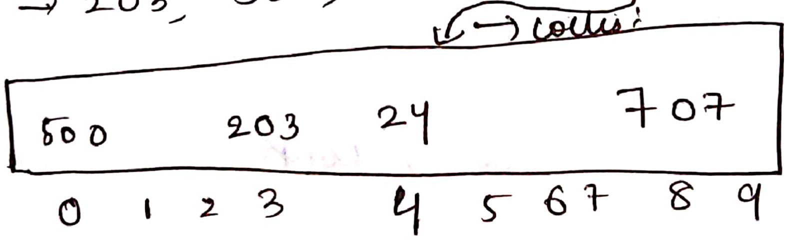


⇒ hashing Solve Space related Problem, but collision resolution is Pending

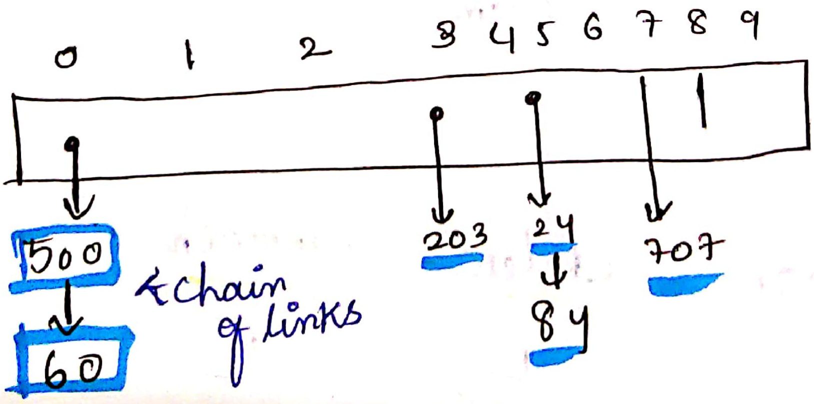
Collision resolution techniques

1) Seperate chaining → linked list, → BBST
2) open Addressing → Linear Probing, → Quadratic Prob, → Double hashing

⇒ 6 → 203, 500, 707, 24, 84, 60



S ⇒ 10



① Linked List

Insert(x) Search(x) delete(x)

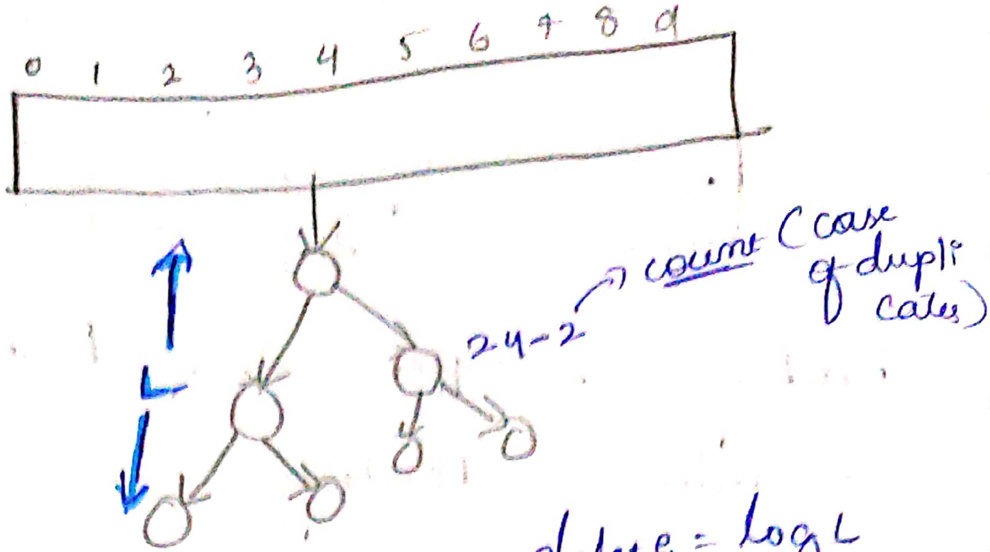
1/ 1 L L

at the head

(2) B.B.S.T (Balanced binary Search)

$H = \log_2 N$

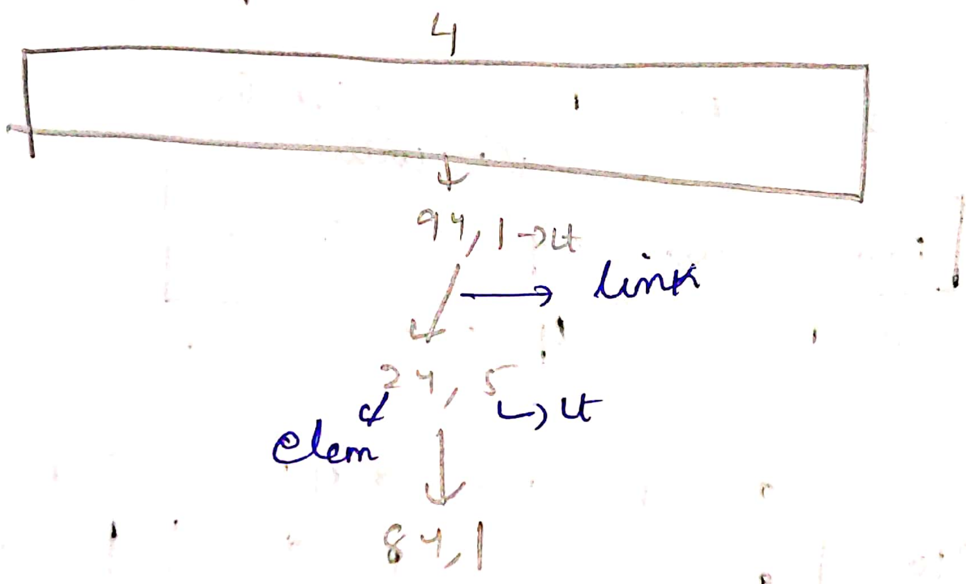
L → is height



Insert ⇒ $\log_2 L$, Search = $\log_2 L$, delete = $\log_2 L$

→ 6 → 203, 500, 707, 24, 84, 60, 94, 24, 24, 24, 24

for multiple elements we can take frea and increment or (duplicates),



⇒

Insert, delete Search

L, L, L

(length of link)

Linear Probing

94

5 ⇒

0, 1 2 3 4 5 6 7 8 9

500		203	24	84	94	707	34

Contents ⇒ 6, 203, 500, 707, 24, 84, 60

$h(x) = (x+i) \% 5 \qquad i \Rightarrow 0, 1, 2, \dots$

$\Rightarrow (203+0) \% 10$

$\Rightarrow (84+0) \% 10 \rightarrow$ occupied

$\Rightarrow (84+1) \% 10 \Rightarrow$ Empty

If occupied it will check for the remaining indexes
If we empty we can ~~Put~~ Store the value.

Quadratic Probing

$h(x) = (x + i^2) \% 5$

34

0, 1 2 3 4 5 6 7 8

				4			60

6, 203, 500 707 24 84

⇒

for $(34+0^2) \% 10$ ⇒ already exist

$(34+1^2) \% 10 \Rightarrow$ already exist

$\Rightarrow (34+2^2) \% 10$

⇒ 8th index

$603 + 0 \rightarrow 3$
$603 + 1^2 \rightarrow 4$
$603 + 2^2 \rightarrow$

LP Vs QP probes

1, 2, 3, 6,

LP: 3 4 5 803, 404, 504, 704

S => 100 → 403, 503, 603, 703, 803,

↓

Size of table

4 4 3,6

QP: 403 4 7 2 3 7

Problem p: 8

Probes 1 2 3 803

$$100 \overline{)703} (70$$
700
3

2

$$100 \overline{)703} (70$$
700
3

= 704
705
706

$703 + (1)^2 \Rightarrow 704$
$703 + (2)^2 = 707$
$703 + (3)^2 = 712$

$$100 \overline{)803} (80$$
800
3

$= 803 + 1^2 =$
$803 + 2^2 = 808$
$803 + 3^2 =$
$803 + 4^2 =$

$$10 \overline{)819} (80$$
800

LP Probes → 0, 1, 2, 3, 4, 4, 5, 6 → 25

QP Probes → 0, 1, 2, 3, 4, 1, 2, 3 → 16
(Jumps)

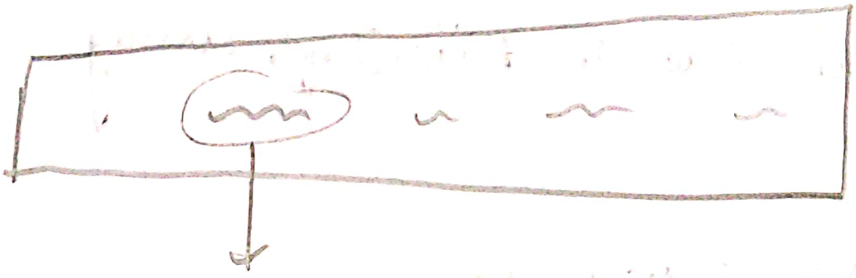
Note: Quadratic Probing is better when compared to lp.

If ~~crystal~~ clusters is huge it will take time to get in to
→ correct position

cluster Size

Lp:

Qp



Smaller
cluster
less no of steps

Cluster Size $\propto$ no of Probes

time complexity

Insert	Search	Delete
P	P	

P= Probes

S ⇒ 10 :

0	1	2	3	4	5	6	7	8	9

nums → 6, 203, 500, 707, 24, 84

Availability

0	1	2	3	4	5	6	7	8
T	T	T	~~T~~	~~T~~	~~T~~	~~T~~	~~T~~	T
			F	F	F	F	F	

True → Empty

F ↓ already Pres

delet → 84 means get the Index and make it is True in Availaple array

Search → 34 ⇒ ~~4~~ Probe
5

you → will think 34 is not Present
Problem is we don't differentiate btwn Empty and delete

→ If deleted keep on Searching
→ If Empty stop Searching & found!

3 things
→ Empty
→ occupied
→ Deleted

⇒ taking Integer array

1) Empty → 0
2) Occupied → 1
3) Deleted → -1

⇒ S=10:

0	1	5
800	203 24	84 34 707

Availibility int

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

deleted

delete → 84 ⇒

Search → 34

3) Double hasing (far cluster)

$$h(x) = h_1(x) + h_2(x)$$

or

$$h_1(x) * h_2(x)$$

some func

$h_1(x) \Rightarrow$ linear $(i^2+x)\%s$

$h_2(x) \Rightarrow$ $(x+i^2)\%s$, $(x+x+i^2)\%s$

Note:- $\{(a^i)\%s$, $(x!)\%s$, $(x^x)\%s\}$ bad

→ Pow(x, i) → log } more time Complexity

Good Hash functions

1) uniformly distribute the keys over a table
2) Easy to compute
3)

Final time Complexity

SC → numbers

OA → 100

OA → 100

OA → 100

OA → 100

table Size

10 → Yes

10 → NO

100 → Yes 0%

1000 → Yes (90% Empty)

10000 → yes (99% empty)

Higher Size more Search optimization

=)

Internal resizing

double

1000 → 200

101 → 200

Class Hashmap {

Int ht [100000];

Int availability[10000] = {0};

// Void Insert

while loop i=>0,

int Val = (x+i) % 10000;

If (availability[value idx] != 1)

ht [value idx] = x;

availability[value] = 1

}

~~bool~~ Void deletion (int x)

int y = Search(x);

If (y! == -1)

avail [y] = -1;

else

Print("element found)

int Search(int x) {

=> loop i=>0, . . .

int ~~Value~~ idx = $(x+i)^2$ / 10000

Empty

If (availability[index] == 0)

ret false

found :-

If (availability[index] == 1 && ht[index] == x)

Return index

```
class HashMap {

    int ht[10000];
    int availability[10000]={0};

  void insert(int x){
    loop i=0; ....
    int index = (x+i^2) % 10000;
    if( availability[index] != 1 ) {
            ht[index] = x;
            availability[index]= 1;

    }
  }

void delete(int x){

   int y= search(x);
   if (y != -1)
       availability[y]=-1;

}

int search(int x){
  → loop i=0; ....

       int index= (x+i^2)% 10000;
       if( availability[index] = = 0)
            return -1;

       if( availability[index]==1 && ht[ind
            return index;
       ....
  }
}

}
```