7. Hashing

29 June 2024 13:59

=> We want to store key-value pairs.

abaacd

a → 3
b → 1
c → 1
a → 1

→ On avg search ≥ O(1)
delete
insert

array → insert → end → O(1)
search → O(n)

Direct Address Table :-

↳ directly use keys as indexes.

0 - 10,000

int c[10001]

insert(10) insert(20)
search(10) insert(30)
delete(5) — O(1)

→ c[10]++;
c[20]++;
c[30]++;
c[10]--;

c[10] ≥ 1 → true
→ false

Problems with DAT :-

① Only use when values are small integers.

val → 0 - 10^{12} → c[$10^{12}+1$] →

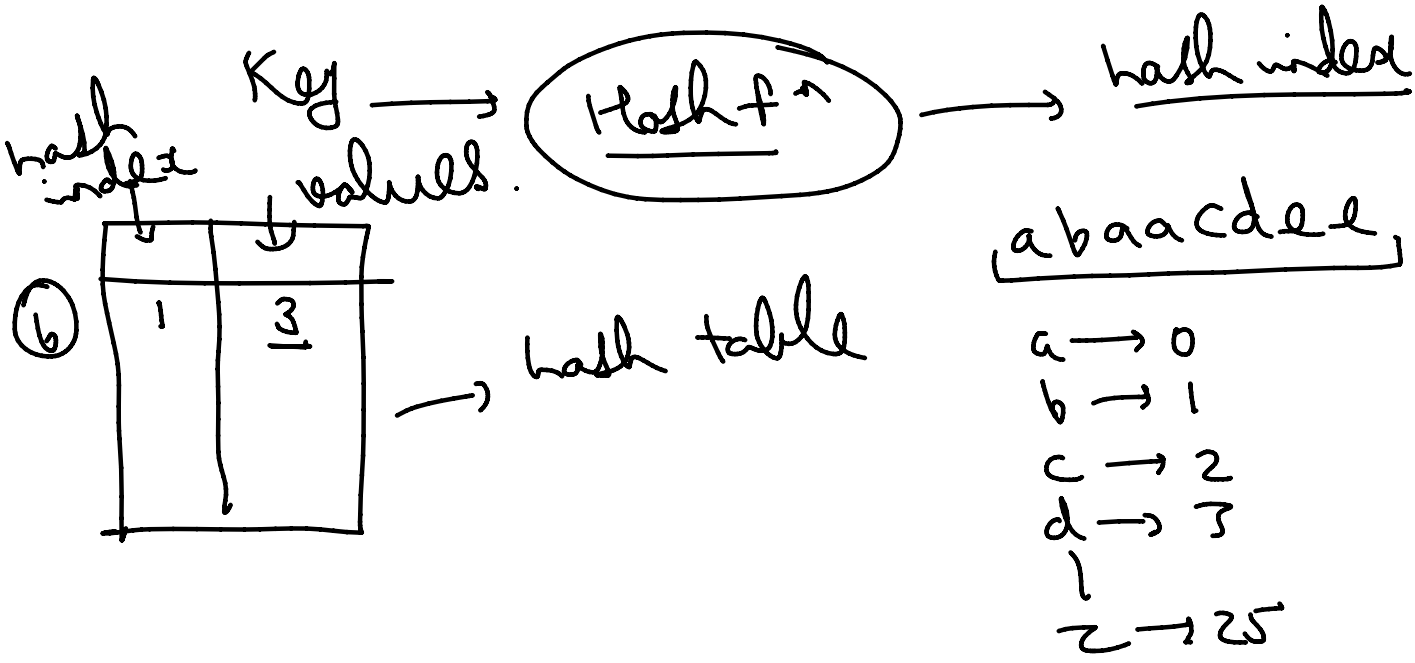
[. .]

val → 0 - 10^{12} → c[10 ...

② Space wastage [10000 - 20,000] c[20001]

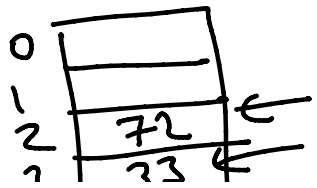
① Cannot use when keys are not int.
eg:- keys → strings
"GFG" → 100

→ Hashing :-
It is a Technique to convert keys into small int & then use that int further for indexing.



→ Hash functions :-
collision.

① h(K) = K%10;
(0-9)



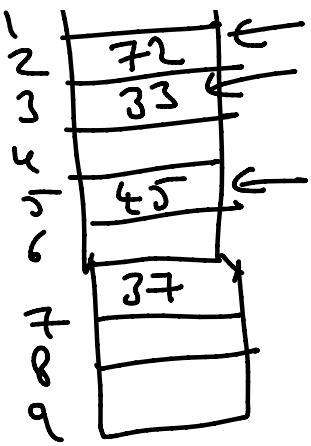
(1) (0-9), -1

i → 72, 37, 45, 33, 43

72%10 = 2

s → 72 → 72%10 = 2

d → 45 → 45%10 = 5



a → 0
b → 1
c → 2

(2) Weighted sum :-

K = abcd

$h(K) = \left((a \times 10^0 + 1) + (b \times 10^1 + 2) + (c \times 10^2 + 3) + (d \times 10^3 + 4)\right) \% m$

bacd ←

m → size of hash table.

Direct sum :- acbd → abcd → 24 → (1+2+3+4) % m

dacb → bcad → (2+3+1+4) % m

⇒ Preferred characteristics of hash fn :-

(1) It should distribute keys uniformly.

(2) Should not be ambiguous. It should return same hash index for same key

(3) It should be fast.

Ideally, integers → constant

strings → O(len)

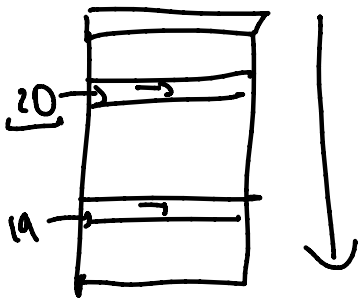
○ strings → O(len)

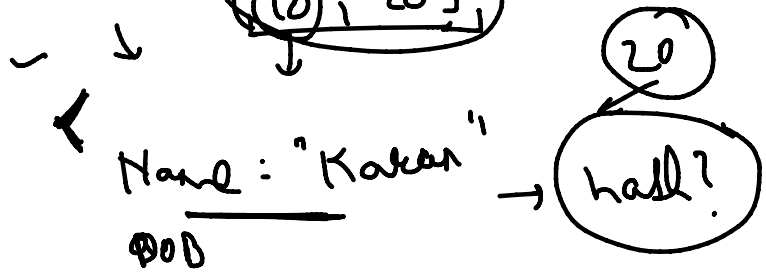
④ If hash table size → m, it should return values 0 to m-1.

C++ → unordered_map
Java → HashMap

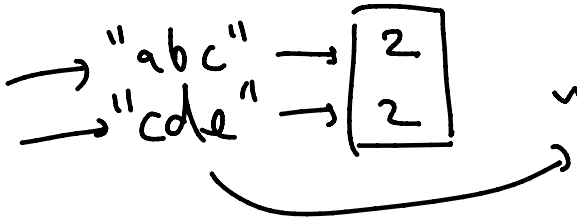
→ Not used when:-

① We are query a range or in sorted order.

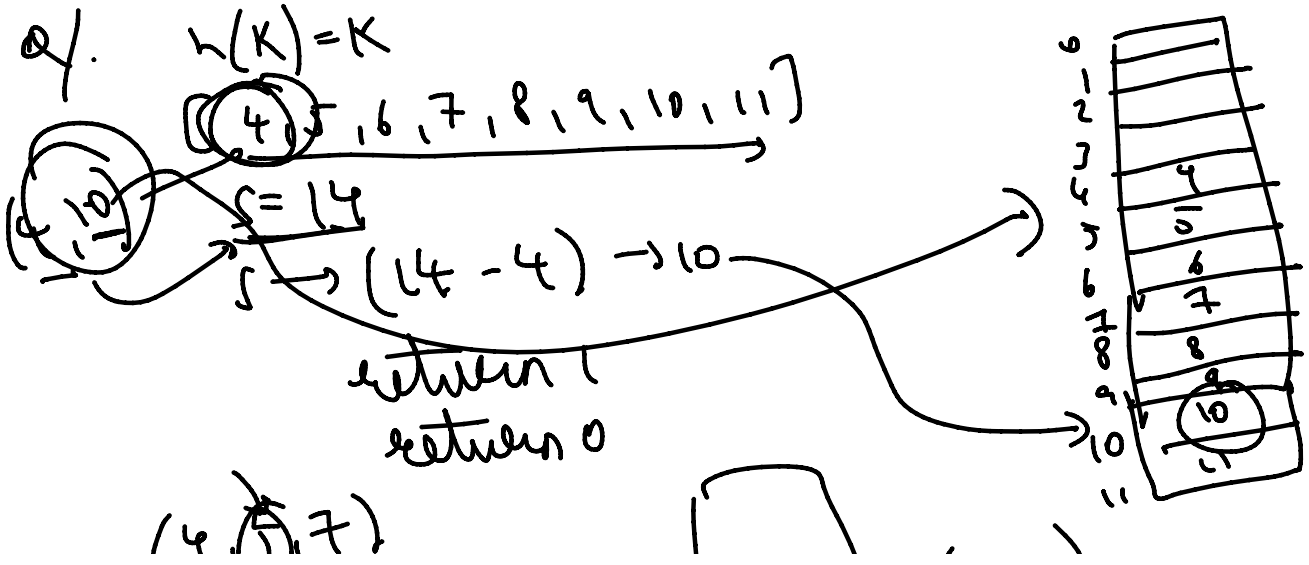


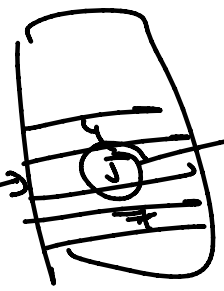


③



Q. $h(K) = K$





(4, 5, 7)

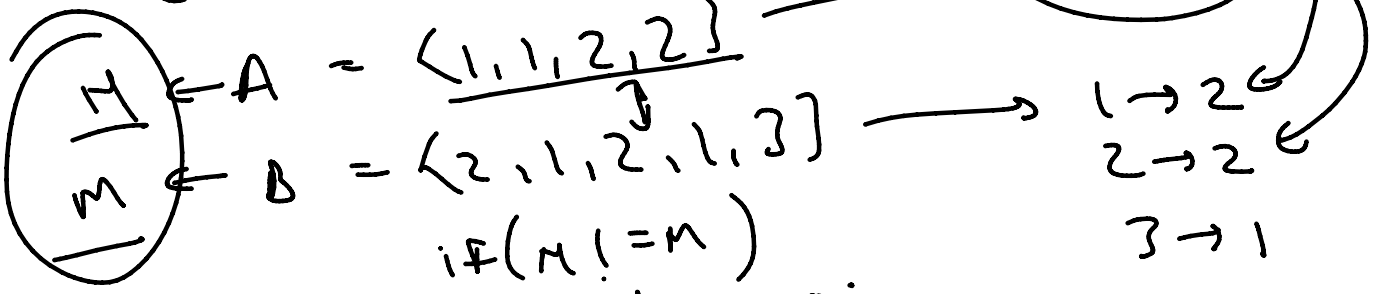
S=10

S-5 = 10-5

(5, 5)

(4, 5, 7) S=10 → 0

(4, 5, 5, 7) S=10 → 1



Q/. a = <1, 3, 1, 2, 2, 5, 5, 5, 5, 4]

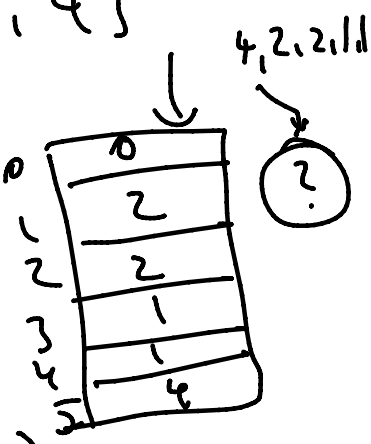
→ 1 → 2

→ 2 → 2

3 → 1

4 → 1

5 → 4



5, 5, 5, 5, 1, 1, 2, 2, 3, 4

Q

(<1, 2], <2, 2], <3, 1], <4, 1], <5, 4)

<4, 1], <2, 1], <2, 2], <1, 3], <1, 4], <2, 5]

sort →

bool comp (int a, int b)

10 5

```
{
    if(a < b)
        return T;
    else
        return F;
}
```

sort(A, A+n, comp);

a = {5, 2, 3, 2, 5, 1, 1, 1} → 1→3, 2→2, 3→1

a' = {<5,2>, <2,2>, <3,1>, <2,2>, <5,2>, <1,3>, <1,3>, <1,3>} → 2

↓ sort

a' = {<1,3>, <1,3>, <1,3>, <2,2>, <2,2>, <5,2>, <5,2>, <3,1>}

(1, 1, 1, 1, 2, 2, 5, 5, 3)

Q/ A1 = [1, 5, 3, 7, 9, 8, 10, 4, 8]

A2 = [8, 9, 7, 3, 10] → (3, 7, 8, 9, 10)

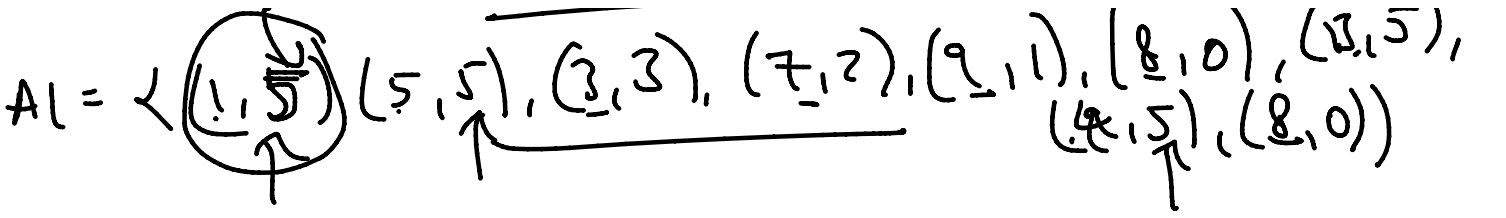
A1 = [8, 8, 9, 7, 3, 1, 4, 5, 13]

A2 = [8, 9, 7, 3, 10]

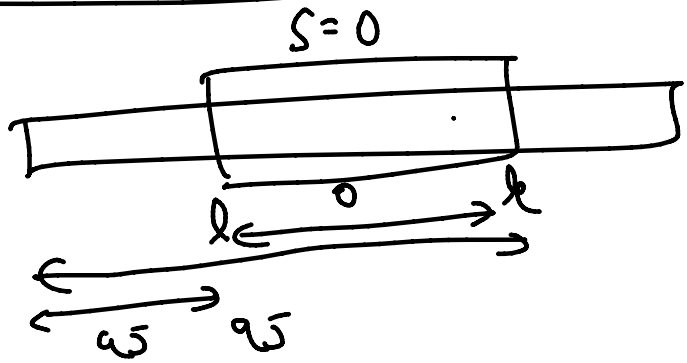
0 1 2 3 4

m[addr[i]] = i;

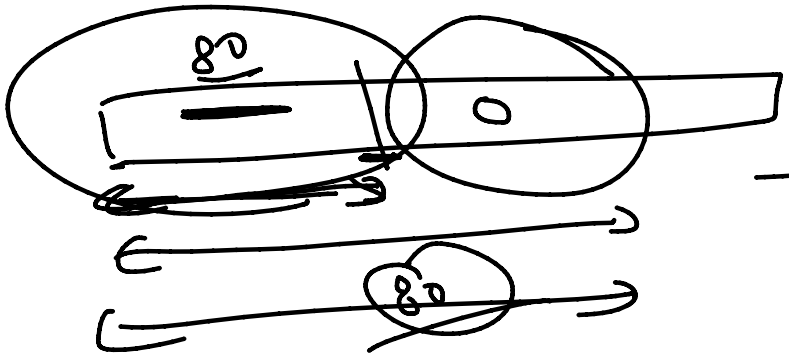
A1 = {(1, 5), (5, 5), (3, 3), (7, 2), (9, 1), (8, 0), (13, 5),



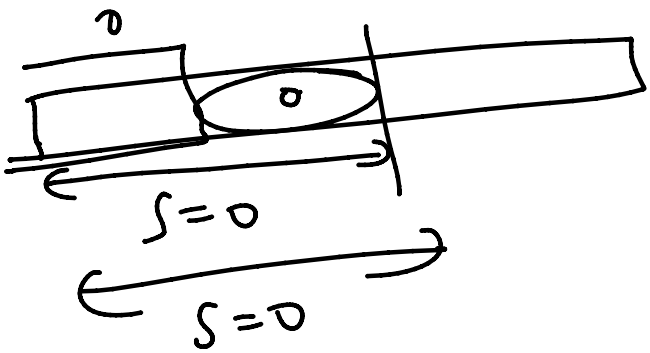
Q.

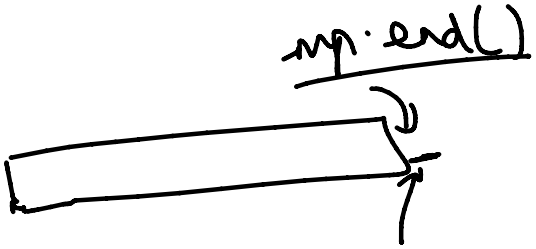


pre_sum[r] = pre_sum[l-1]



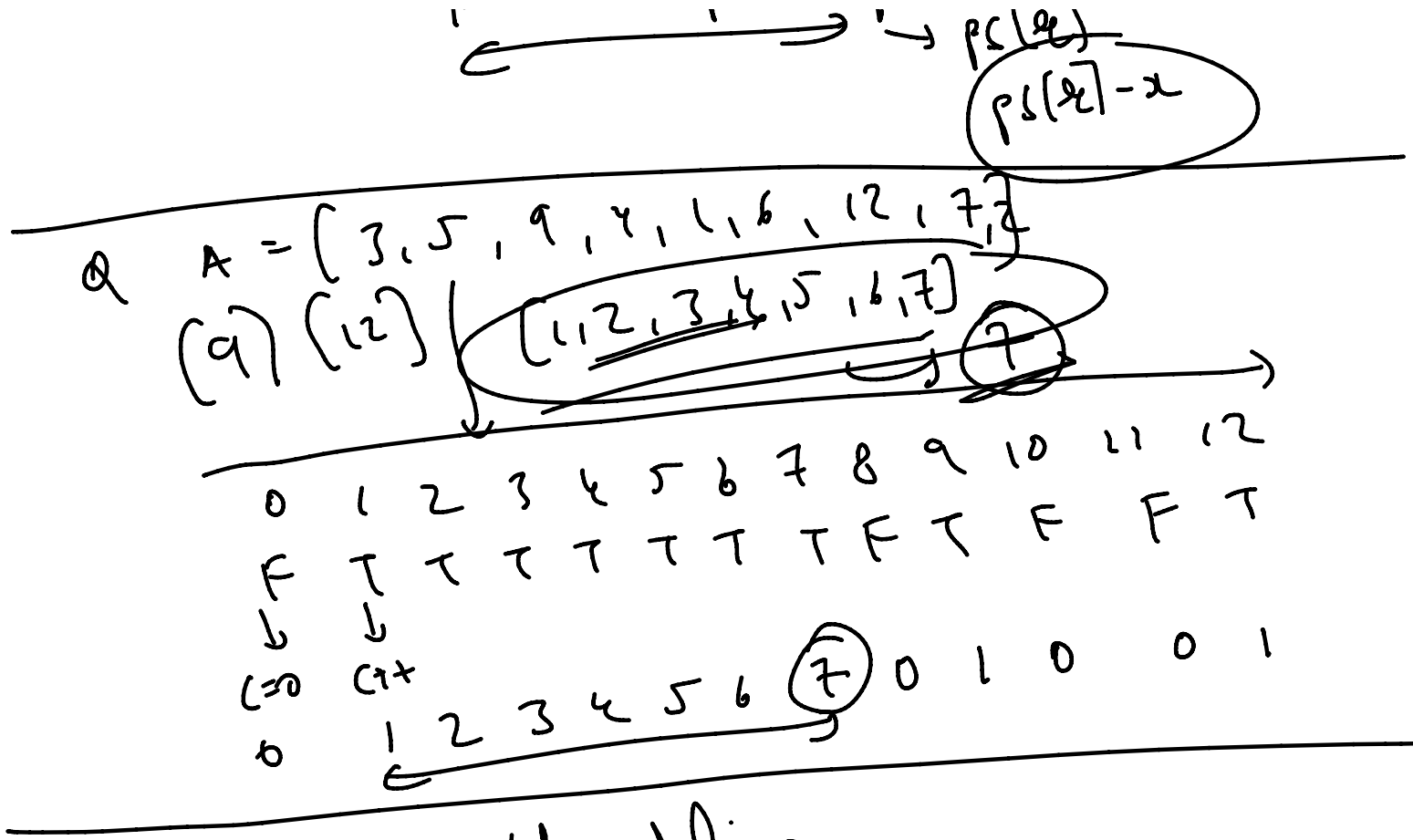
mp[80] = third





Q.

pres[l-1] + $\frac{x}{0}$ = pres[r]



⇒ Collision Handling :-

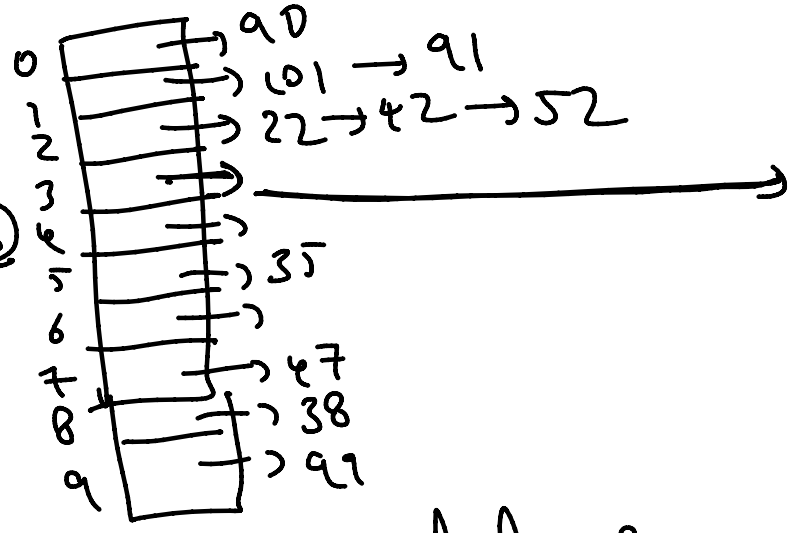
→ Chaining

→ Open Addressing

① Chaining :-

$h(K) = K\%10;$

i → 22, 35, 47, 90, 42, 38, 99, 101, 91, 52



Load factor :- → no. of keys

Load factor :-

$\alpha = \frac{n}{m}$

$n \rightarrow$ no. of keys

$m \rightarrow$ hash table size

assuming uniform distribution

→ Chaining → Linked List, vector :-

insert → $O(1) / O(1+\alpha)$

delete → $O(1+\alpha)$

search → $O(1+\alpha)$

→ Chaining → maintain AVL (Balanced BST) tree for each node.