Hash Maps

what ?

- This is an data structure
- that gives insertion, deletion, searching in O(1) time.

why ? :-

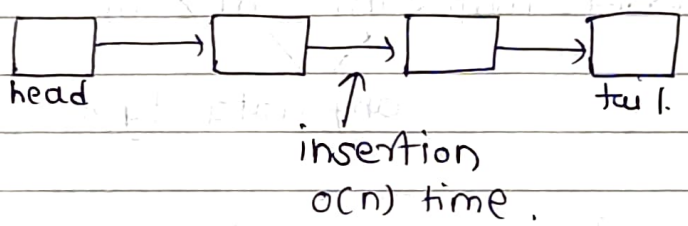
I/P → "Ek word tha Ek word word tha"

O/P : gives maximum occurance of word with O(1) time ?? [expect traverse the loop]

That's why maps comes in picture.

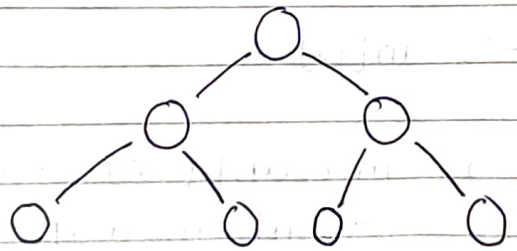
Implementation :-

① using Linked list.



deletion and searching also take O(n) time.

② using BST.



Insertion → O(log n)
Deletion → O(log n)
Searching → O(log n)

∴ using LL & BST is not possible to insert, delete, searching in O(1) time.

Then we have,

① unordered_map → O(1) time.

② map → Implement using BST
↳ Insert, Delete, searching done in O(log n) time.

Unordered_map Implementation :

① STL

```
# include <unordered_map>

unordered_map <int, int> m
                  any data type.
```

i) insert → pair<int, int> P = new_pair(10, 20)
m.insert(P).

→ pair<int, int> P (10, 20)
m.insert(P).

→ m[10] = 20.

ii) count → m.count(30) → if key is not present in map gives '0'

m.count(10) → gives '1'.

iii) At → if we try to get key which is not present in map.

e.g. map

<10,20>
<30,40>
<50,60>

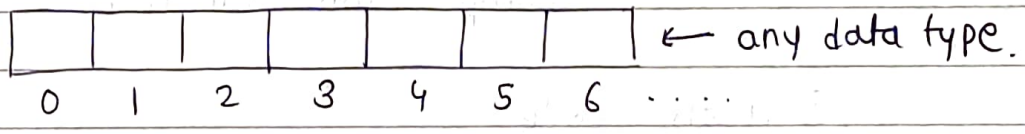
- if we write m[90] to be printed, which is not present in map.
- It creates an entry of m[90] in map & assign value to '0'.

IMP * m.at(80) → 80 key also not present in map.

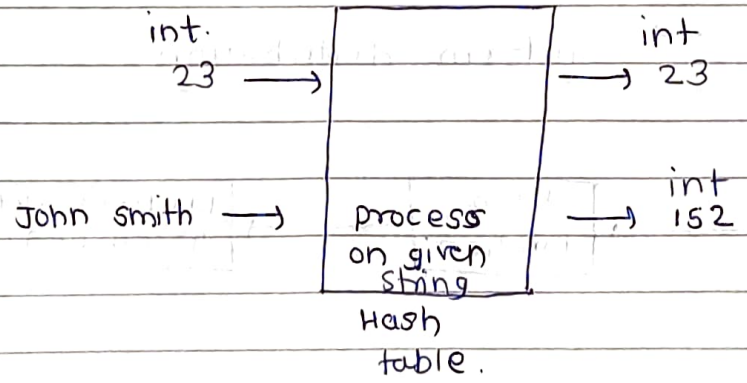
→ This gives error & abort the program

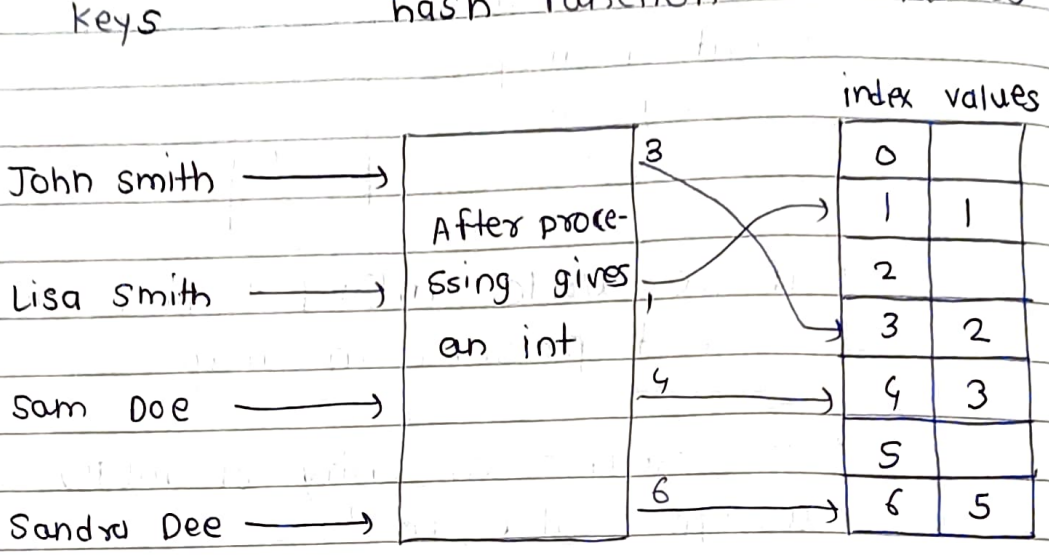
② Implementation of map using Hash tables.

- we have an Bucket array



- Also have Hash table





∴ we get

m["John smith"] = 2

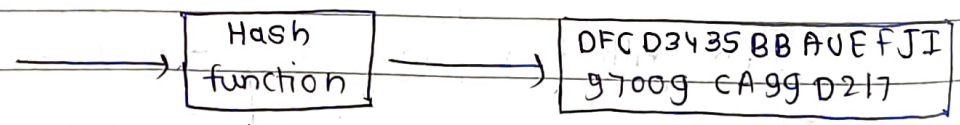
* Hash function.

- Hash code
- compression function.

① Hash code.

why ? → conversion of data type to int.

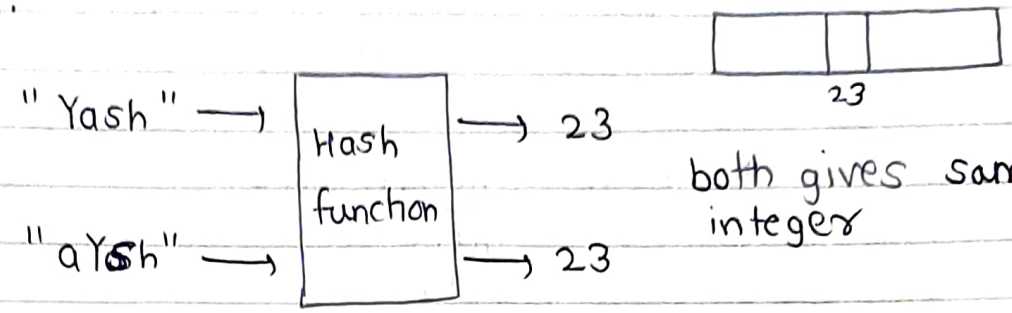
→ uniform distribution.



- collision resolution :

what is collision in Hash table :

e.g.

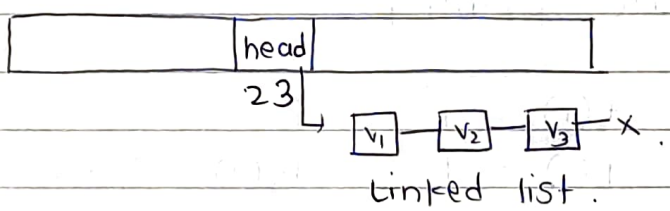


both have point to same index in an array This is collision.

So, collision handling comes to picture.

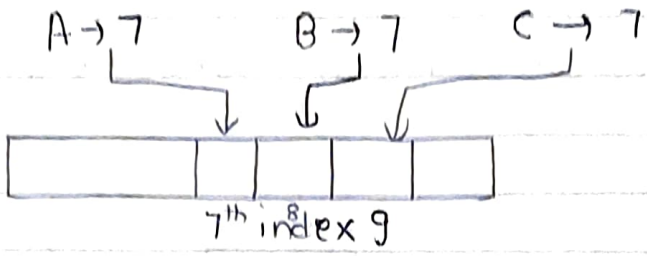
① open hashing (seperate chaining).

"Yash" & "aysh" both giving 23 index.



BUT, if 100 words are pointing to same index. This open hashing is not possible, it takes time to insert, delete, searching.

② open Addressing.



→ $H_i(a) = \underbrace{h(a)}_{\text{index } (7^{th})} + f_i(a)$.

$f_i(a)$ can have multiple probing.

① linear probing.

$f(i) \rightarrow i$
$f(1) \rightarrow 1$ $\quad H_i(a) = 7 + 1 = 8$
$f(2) \rightarrow 2$ $\quad H_i(a) = 7 + 2 = 9$.

② Quadratic probing.

$f(i) = i^2$
$f(1) = 1$ $\quad H_i(a) = 7 + 1 = 8$
$f(2) = 4$ $\quad H_i(a) = 7 + 4 = 11^{th}$ index.

③ we can implement our own probing also

e.g. $i^3 + 2$, $i^3 + 13$, $i^4 + 9$.

* complexity analysis.

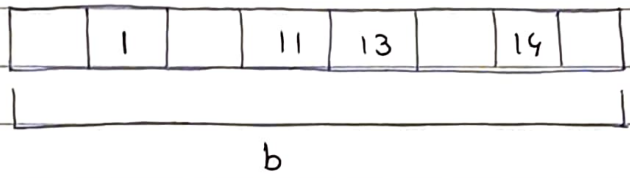
eg. "merq bhai love babbar bro kya haal"

n → total number of words.
k → words length.

To find or convert word to int it will O(k). but we assume that n >> k (n is much larger than k).

∴ k will ignored T.C = O(1).

*



n → no. of entries
b → no. of boxes available.

we always ensure $\frac{n}{b} < 0.7$ → load factor.

when $\frac{n}{b} < 0.7$ it is very safe to ensure T.C = O(1)

if number of entries are increasing, then we will increase the bucket size by 2x.

∴ $\frac{n}{2b} = 0.5x$ → increased by half.