Hashing

↓

pre-storing & fetching when required

→ Why hashing?

Suppose given an array | 1 | 2 | 1 | 3 | 4 | and we need to return the count of some no. let's say 1, 2, 4, 10.

The very first approach would be linear search.

$4 * O(N)$

What if $N = 10^5$ & no. $= 10^5$

$O(10^5 \times 10^5) = 10^{10} \simeq$ 100s to execute.

→ Solution: Let's say it's given that the max value of array can be 12.

So we make a hashArray / freqArray (precalculation)

| 1 | 2 | 3 | 4 | 1 |

↓

We have stored the count.

Now, count of 2 = hash[2] = 1 in O(1)

→ What if max element of arr is 10^9?

arr[1e9+1] → Segmentation fault.

int arr[1e6] → inside main — bool 1e7

int arr[1e7] → global — 1e8

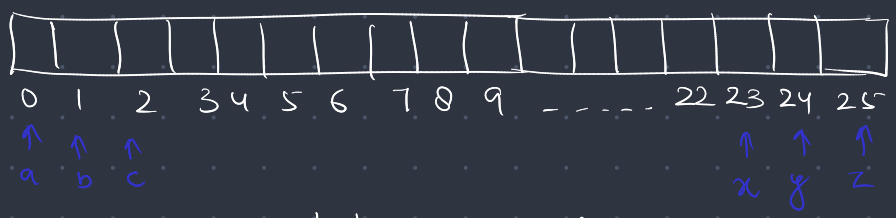
→ Character hashing :-

S = "abcdabefe"

a → 2
c → 2
z → 0

↳ Only lowercase.

26 elements.



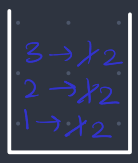
'a' - 'a' = 0
'b' - 'a' = 1

ch - 'a'

→ Solution of number hashing :-

map & unordered_map

arr = 1, 2, 3, 1, 3, 2



takes only size = unique elements inside array.

map < int, int >
key freq

→ Time complexity : O(log N) { Store & fetch } Both { best, worst, avg. } map

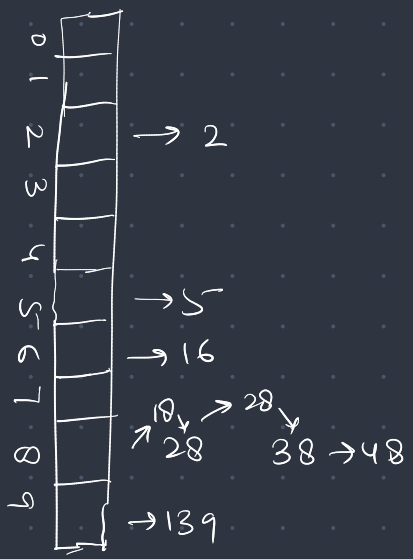
→ Time complexity : O(1) { Store & fetch } Both { best, avg. } unordered
O(N) — worst

→ First preference to be given to 'unordered_map' then go to map if TLE is encountered due to internal collision.

→ Under the hood?

- ↳ Division Method
- ↳ Folding Method
- ↳ Mod Square Method

→ 2, 5, 16, 28, 139 ~~>10~~ arr[i] % 10



2, 5, 16, 28, 139, 38, 48, 28, 18

→ All the keys, ends up to happen at the same key. which result in O(N). This is collision.

Eg: 2, 22, 12, 222, ---- 243462

Hash function

- Hash code
  - ↙ Conversion to int so that we can map
- Compression
  - ↳ Takes the value (int obtained from hash code) in the range.

→ Hash Code :-

Identity fn

23 → [H.C] → 23

"Shubham" → [H.C.] → 23

→ Collision Handling :-

↳ Open hashing → Same place pe hi jao.

↳ Open addressing → Already pada hai toh doosre jagah pe dekh lo na.

$H_i(a) = h(a) + F_i(a)$

↓

ith attempt me kahan par place krna hai.

Linear probing →

$F(i) \rightarrow i \Rightarrow H_1(a) = h(a) + 1$

Quadratic probing → $F(i) \rightarrow i^2$

⇒ Let n → no. of entries & b → no. of boxes available.

No. of entries in a box = $n/b < 0.7$ → load factor

We need to maintain this so that our hash fn works.

→ Rehashing :-

$n \uparrow \Rightarrow$ We need to increase b.

↓

Rehashing (increase bucket size)