NOTES:

HASHMAP

Good Morning

Note :- Concept will be correct

Syntax → will share a doc

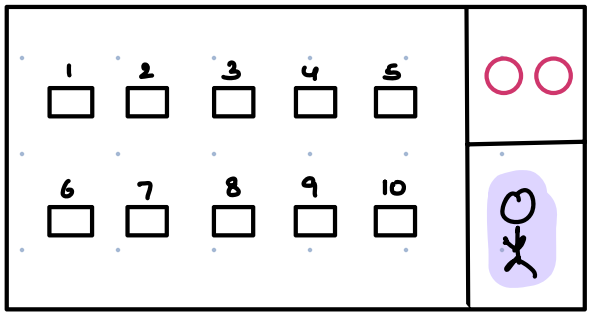
Today's Agenda

- → Hashmap Introduction
- → Hashset Introduction
- → Frequency of each query
- → First non repeating ele
- → distinct elements
- → Subarray with sum = 0 (Interview)

* How things are implemented in HashMap ↓

Hashmap implementation

Yogi



Register

1 → Occupied

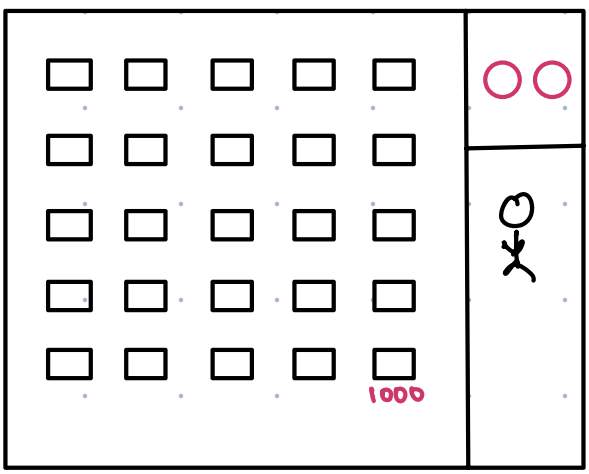
2 → Not occupied

3 → Occupied

⋮

10 → Occupied

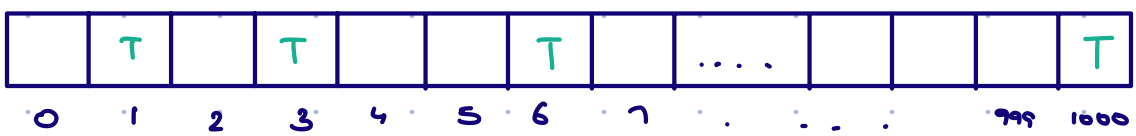
1000 rooms



Room no. → {1 1000}

boolean array [ ] = 1001

Status [ ] =



status array

True → Room is occupied

False → Room is available

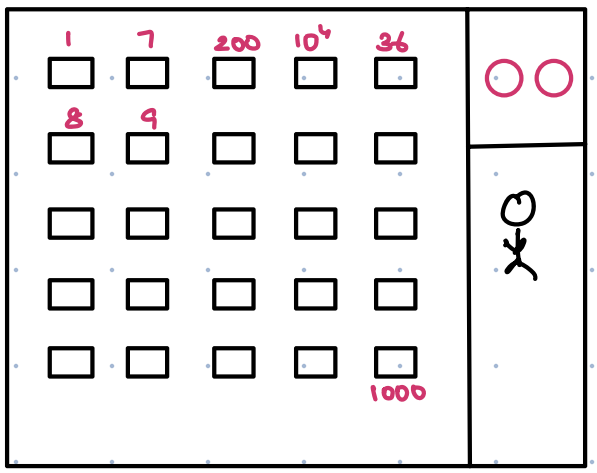
To check if room is available or not?

TC: O(1)

* Covid → Hotel Business is not doing great.

Numerologist ⟶ [1 to 10^9]

1000 rooms



One on one mapping

boolean [] status = $10^9 + 1$

Advantage → Access the room status → TC: O(1)

Issue → For 1000 rooms

↳ space of $10^9 + 1$

consuming a lot of extra space to store information of 1000 rooms

Hashmap → A DS which holds the information in

key - value pairs

Hashmap<Key, Value>

Room no. T/F

10^4 → True
8 → True
9 → True

max size(hm) = 1000

any room

hm[9] = True

room 9 is occupied

check the value for a particular room → TC : O(1)

* Implementation of HM & its working → covered later in classes

* keys → Always be unique
  Value → Can be anything

01. Store population of every country

key : County name {String}

Value : Population {long}

HM <String, long>

02 No. of states for each country

Key : County name {String}

Value : No. of states {Integer}

HM < String, Integer >

03. Name of all states of each country

Key : County name {String}

Value : Name of all states { AL<String> }

HM <String, AL<String>>

04. Store population of each state in every country

Key : County name {String}

Value : population of each state : {HM<String, Long>}

HM < String, HM<String, Long>>

Hashmap → To store key-value pairs

HashSet → To store unique keys

HashMap	Hashset
01. size() → No. of keys in HM	01. size() → No. of keys in hashset
02. insert(key, value)	02 add(key)
03. search(key) → T/F	03. search(key) → T/F
04. remove(key)	04 remove(key)
05. update → insert(key, value)	05 Iterate on hashset ↓ to get all keys
06. get(key) → Fetch the value for this key	
07. Iterate on hm ↳ get all the keys	

A single operation in HashMap / Hashset is

TC: O(1)

* India → 20
* USA → 23
* Pak → 21
* Swi → 17
* France → 4
* India → 4

HashMap<String, Integer>

```
hm.insert(India, 20);
hm.insert(USA, 23);
hm.insert(Pak, 21);
hm.insert(Swi, 17);
hm.insert(France, 3);
```

hm. insert (India, 4);

HM

USA	→	23
Swi	→	17
India	→	4
Pak	→	21
France	→	4

TC : O(n)
SC : O(n)
} To insert n entries

* Ordering inside hashmap will not be same as ordering of insertion.

* If two duplicate keys are inserted, then there will be only one key, with the modified value. (Overwrite the previous value)

7:56 AM → 8:06 AM

Hashing Library Names in Different Languages

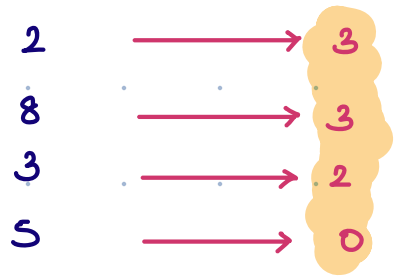
	Java	C++	Python	Js	C#
Hashmap	Hashmap	unordered_map	dictionary	map	dictionary
Hashset	Hashset	unordered_set	set	set	Hashset

Q → **Find frequency of numbers**

Given N array elements & Q queries, find frequency of elements (given in queries) from the array.

ar[10] = {2 6 3 8 2 8 2 3 8 10}

Queries: 4



Brute force → for every query, iterate on the array & count its frequency.

TC: O(Q*n)

SC: O(1)

02 Sort the array

TC: O(nlogn + Q*n)

SC: O(1)

03. Optimised Approach - Use Hashmap

Key → Distinct ele

Value → Freq of distinct ele

ar[10] = {2 6 3 8 2 8 2 3 8 10}

HM<Integer, Integer> map

Key		Value
2	→	x x 3
6	→	1
3	→	x 2
8	→	x x 3
10	→	1

Queries

2 → map.get(2)

8 → map.get(8)

3 → map.get(3)

5 → map.get(5)

```
void printquery ( int [] A, int [] Q)

    HashMap < Integer, Integer> hm;

    // Iterate on array & populate HM

    for (i = 0 ; i < n ; i++){

        if ( hm.search ( A[i]) == = True ){

            int of = hm.get (A[i]);

            int nf = of +1

            hm. insert (A[i], nf )

        }

        else {

            hm. insert (A[i], 1) ;

        }

    }
```

```
    // Iterate on queries
    for (i=0; i<Q.length; i++){
        int ele= Q[i];
        if (hm.search(ele) == True){
            print(hm.get(ele));
        }
        else   print(0);
    }
}
```

TC: O(Q+n)

SC: O(n)

02 Find the first non repeating ele.

↳ first ele which is not repeating from start

A[] = {4 3 3 2 5 6 4 5} Ans = 2

A[] = {1 3 2 1 4} Ans = 3

A[] = {2 3 4 1} Ans = 2

BF → For every element, iterate & look for its frequency. If freq == 1 → return ans.

TC : $O(n^2)$

SC : $O(1)$

* Optimal Approach → To use hashmap

01. Build hashmap

02. Iterate on array & return first ele with freq as 1

A[] = { 1 4 3 6 3 4 1 7 3 }

HM =

Key	value
7	1
1	2
4	2
3	3
6	1

* Code → {TODO}

03. Count of distinct ele.

A[] = { 2 3 1 3 4 } Ans = 4

A[] = { 3 3 3 } Ans = 1

* Idea → Use hashset & insert arr elements inside it.

```
HashSet<Integer> set;

for (i=0; i<n; i++) {
    set.add(A[i]);
}

return set.size();
```

TC : O(n)
SC : O(n)

Qs. Given ar[N] elements, check if there exists a subarr with sum = 0

ar[10] =

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

Ans = True

Brute force → Consider every subarray, & check if sum == 0 or not.

3 nested loops	Prefix sum	Carry forward
TC : $O(n^3)$	TC : $O(n^2)$	TC : $O(n^2)$
SC : O(1)	SC : O(n)	SC : O(1)

* Optimisation using prefix sum

ar[10] =

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

Pf [10] =

2	4	5	2	6	9	10	8	5	7

$$sum[i\ j] = psum[j] - psum[i-1] = 0$$

$$psum[j] = psum[i-1]$$

$$pf[2] = 5 = sum[0-2]$$

$$pf[8] = 5 = sum[0-8]$$

$$sum[3-8] = pf[8] - pf[2]$$

$$= 5 - 5$$

$$= 0$$

Conclusion → If we have a value repeating in our psum array, then we have a subarr with sum = 0

$$A[\,] = \{2 \quad -5 \quad 3 \quad 6\}$$

$$pf[\,] = \{2 \quad -3 \quad 0 \quad 6\}$$

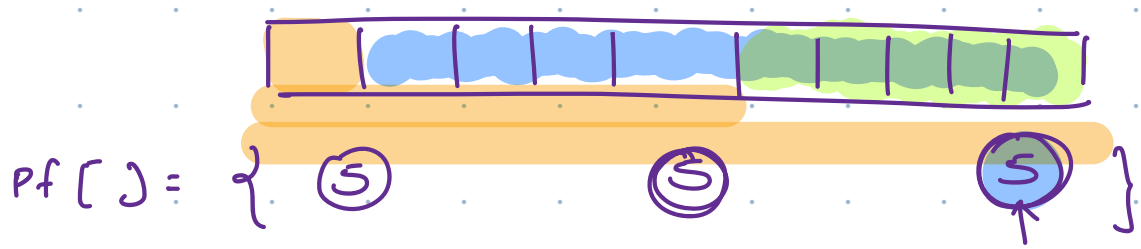
Note - If 0 is present in pfsum[], we have a subarr with sum = 0

1st way = `if (pf[i] == 0) return true;`

2nd way = Explicitly add a 0 by yourself.

```
boolean subarrzero ( int [] arr ){
    int pf [n] ; ⟶ {TODO}
    HashSet <I> set ;
      set. add (0) ·
    for (i=0; i<n ; i++){
        if ( set. search (pf [i]) == True) return true;
        set . add (pf [i]);
    }
}
```

Count



5 → ✗ 2 ans = 0 + 1 + (2)

→ Use hashmap

→ Edge case - map. insert (0, 1)

→ ans = ans + hm. get (pf [i]) ;