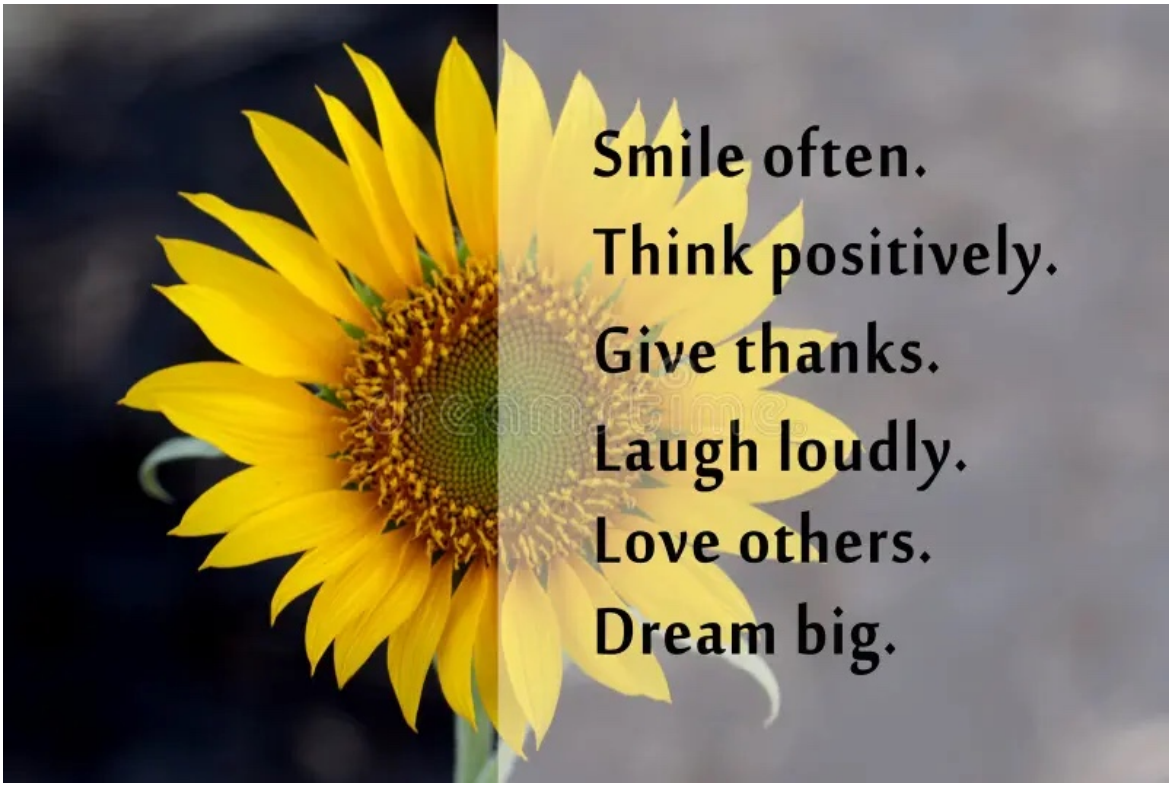
Agenda

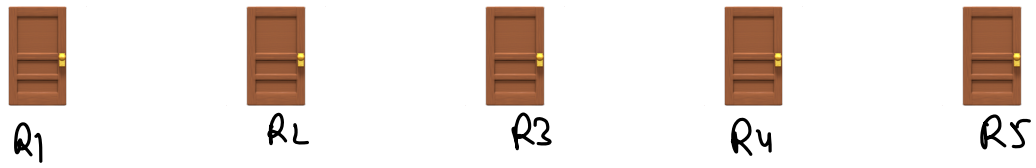
1. Introduction to HashMap & HashSet
2. Given queries, find frequency of an element
3. No.of Distinct Elements
4. Longest Substring Without Repeat



Hotel

R1 →	Occ.
R2 →	Occ
R3 →	~~N.O~~ Occ.
R4 →	N.O
R5 →	~~Occ.~~ N.O

5 rooms



↓

1000 rooms- [0-999]

→ boolean arr [1000]

[1 → 10^9]

5, 25, 35, 15, 45, 55, - -

—— 5555, 5556, - -

—— 88888555

1000 rooms

room no.	→	status
5588	→	Occ.
25	→	N.O
35	→	N.O
95	→	Occ.
5	→	N.O
9955	→	Occ

boolean arr [10^9]

Hashmap / map / dictionary

1. Population of every country

🇮🇳 India → 145

🇺🇸 USA → 65

🇷🇺 Russia → 14

🇨🇳 China → 147

HashMap < String , Long > map ;

2. Number of states of every country

🇮🇳 India → 29

🇺🇸 USA → 50

🇨🇳 China → 25

🇷🇺 Russia → 21

Hashmap < String , integer > map;

3. Name of all the states of every country

🇮🇳 India → Andhra Pradesh, Arunachal Pradesh, U.P, M.P, Karnataka,....

🇺🇸 USA → New York, Washington, Texas,...

🇷🇺 Russia → Moscow, Kazan, Samara,...

🇨🇳 China → X, Y, Z,...

Hashmap <String, list<string>> map;

4. Population of each state in each country

🇮🇳 **India**

Karnataka → 50
Maharashtra → 72
Himachal Pradesh → 36

🇺🇸 **USA**

Texas → 18
Florida → 16
Washington → 32

🇷🇺 **Russia**

Kazan → 3
Samara → 5
Moscow → 7

Hashmap < String, Hashmap < String, Long > > map

Keys must be unique. Values can be anything.

Keys → Integer, Long, String, Double, Float.

Hash - Map (key. value pairs)	Hash - Set (Keys)
→ add (key, value) → size () → remove (key) → update (key, newvalue) → search (key) → get (key) O(1)	→ add (key) → remove (key) → size () → search (key) O(1)

	Java	C++	Python	JS	C#
Hash - Map	HashMap	un-orderedmap	dictionary	map	dictionary
Hash - Set	Hashset	unordered-set	set	set	hashset.

< Question > : Given N elements & Q queries. Find the frequency of elements provided in the query.

$(1 \le N \le 10^6)$

$(1 \le Q \le 10^5)$

N = 11

arr → [2 6 3 8 2 8 2 3 8 10 6]

0 1 2 3 4 5 6 7 8 9 10

Q = 4

2 → 3

8 → 3

3 → 2

BF Idea For every query, iterate on all array elements and find how many times that element is present.

T.C → O(Q×N)
S.C → O(1)

Idea -2 Create frequency map.

arr → [2 6 3 8 2 8 2 3 8 10 6]

0 1 2 3 4 5 6 7 8 9 10

int → int

2	→	3
6	→	2
3	→	2
8	→	3
10	→	1

fmap

Q = 4

2 → 3

8 → 3

3 → 2

< / > Code

```
Hashmap< integer, integer>  map;

for( i=0; i< N; i++){
        if( map.search(arr[i]) == false){
                map.add( arr[i], 1);
        }
        else{
                val = map.get(arr[i])
                map.update( arr[i], val+1);
        }
}

for( i=0; i< Q; i++){
        element = Queries[i];
        if( map.search(element) == false){
                print("0");
        }
        else{
                print( map.get(element));
        }
}
```

T.C → O(N+Q)
S.C → O(N)

< Question > : Count of distinct elements

N = 5 [3 5 6 5 4] ans = 4

N = 3 [3 3 3] ans = 1

N = 5 [1 1 1 2 2] ans = 2

< / > Code

```
Hashset< integer set;

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

return set.size();
```

T.C → O(N)
S.C → O(N)

Determine the "GOOD"ness of a given string A, where the "GOOD"ness is defined by the length of the longest substring that contains no repeating characters. The greater the length of this unique-character substring, the higher the "GOOD"ness of the string.

Your task is to return an integer representing the "GOOD"ness of string A.

Note: The solution should be achieved in O(N) time complexity, where N is the length of the string.

Input 1: A = "abcabcbb"
Output 1: 3

Input 2: A = "AaaA"
Output 2: 2

Eg → pwwkewwt

ans=3

< Question > : Longest Substring without Repeating Characters →

BF Idea

Consider all the substrings and for every substring check for the repeating characters.

T.C → $O(N^3)$

Optimised Approach → Acquire & Release

Str. a b c d m d c m o p q r m m

0 1 2 3 4 5 6 7 8 9 10 11 12 13

i

j

Set

~~c~~
~~o~~
~~p~~
~~d~~
m
~~q~~
~~r~~

maxlen → ~~0~~ ~~1~~
~~2~~
~~4~~
~~8~~ ~~6~~ 7

currlen → ~~1~~ ~~2~~ ~~3~~ ~~4~~ ~~5~~ ~~2~~ ~~3~~
~~4~~ ~~5~~ ~~6~~ ~~7~~ 1

Code →

```
Hashset<Character> set;

maxlength = 1 , j = 0;

for( i=0; i<N; i++){
        char ch = str[i];
        if( set.search(ch) == true){
                while( str[j] != ch){
                        set.remove( str[j]);
                        j++;
                }
                set.remove( str[j]);
                j++;
        }
        set.add( ch);
        maxlength = Math.max( maxlength, i-j+1)
}

return maxlength;
```

[T.C → O(N)
S.C → O(N)]

[Break → 8:36 → 8:44]

Q Given N elements. Find the first non-repeating element.

If non-repeating element is not present, return -1.

N=6 arr → [1 2 3 1 2 5] ans = 3

0 1 2 3 4 5

arr[] - [4 3 3 2 5 6 4 5] ans = 2

0 1 2 3 4 5 6 7

arr[] - [4 1 1 4] ans = -1

0 1 2 3

B.f → for every element, iterate on whole array and check if it is repeating or not.

[T.C → $O(N^2)$
S.C → $O(1)$]

idea-2. → Create fmap

Iterate on array elements & check their frequency.

Code →

```
Hashmap< integer, integer>  map;

for( i=0; i< N; i++){
        if( map.search(arr[i]) == false){
                map.add( arr[i], 1);
        }
        else{
                val = map.get(arr[i])
                map.update( arr[i], val+1);
        }
}

for( i=0; i< N; i++){
        if( map.get(arr[i]) == 1){
                return arr[i];
        }
}
return -1;
```

T.C → O(N)
S.C → O(N)

arr[] - [5 2 7 3 6]

pgcd[] -> [ ]

sgcd[] -> [ ]

Keys will be present in random order in hashmap.