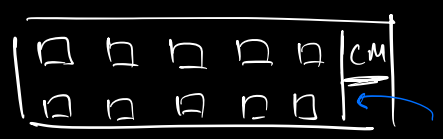
⇒ Hashing :- story

Chandler Monica

↓

started Hotel business

↓



Jainik

& maintain register

↓ expand

10000 rooms

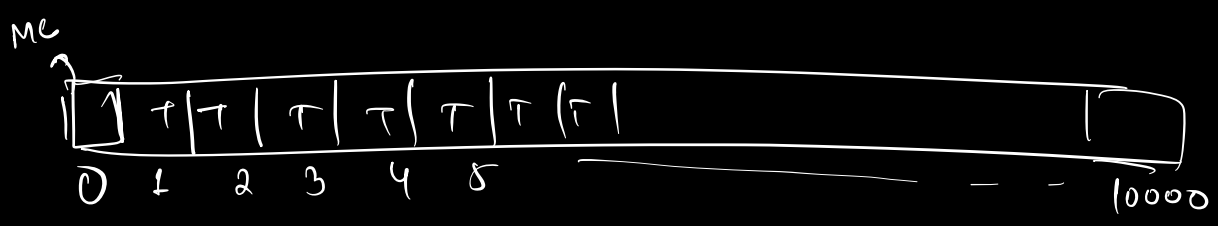
room number [1-10000]

↓

✗ maintain register 10k room ✗

Jainik

boolean array [10001] ⇒ T {available}



arr[5] ⇒ F

room number 10 ⇒ (arr[10] == true)

↓

available.

Pandemic

↓

Phoebe ⇒ room ⇒ lucky nos.

$[1 - 10^9]$

range of room nos. ⇒ $[0 - 10^9]$ ⇒ $10^9 + 1$ rooms nos.

Finite

arr[$10^9 + 1$]

F	F	T	F	T	F	T	F	T	F

↓

$10^9 + 1$ ⇒ lot of waste space

book / check availability ⇒ O(1)

optimisation on space ⇒ ✓✓

⟨Hashmap⟩

* **Hashmap**

<key, value> pairs

↓

key is always unique

Array ⇒ 400, 79, 11, 6, 666, 10543

<400, T> <666, T>

<79, F> <10543, F>.

<6, T>

<11, T>

* Hashset

<key>

↳ keys are all unique.

Q 1.) Store population of every country:-

country → population

HashMap<key, value>

↓ country (String), ↓ population (long)

HashMap<String, long>

Q2. Store no. of states of a given country

HashMap < key, value >

key → country name → String

value → no. of states → Int

→ HashMap < String, INT >

Q3. For every country we want to store all state name.

HashMap < key, value >

key → country

value → List < String >

→ HashMap < String, List < String > >

Q4. For every country, store population of each state

India → Bihar → x, WB → y, UP → z

key: Bihar, WB, UP

value: x, y, z — long

value → HashMap < String, long >

India → String (key)

→ HashMap < String, HashMap < String, long > >

// value ⇒ can have any datatype

// key ⇒ { string, $\underbrace{\text{int, float, long}}_{\text{all primitive + string}}$ }

// HashMap

<key, value> pair

$\underline{\underline{O(1)}}$ TC

- insert (key, value) → inserts <k,v> pair in hashmap
- search (key) → returns value for that key
- remove (key) → remove <k,v> pair
- update (key, newvalue) → updates value of key with newvalue
- size ( ) → no. of <k,v> pairs present

avg. case or amortised case

10^4 operations ⇒ $\underline{\underline{O(1)}}$ avg.

// HashSet

<key> → unique keys

$\underline{\underline{O(1)}}$

- insert (key)
- search (key)
- remove (key)
- size ( )

	Java	C++	Python	JS	C#
Hashmap	Hashmap	unordered-map	Dict	–	dict
Hashset	Hashset	unordered-set	Set	–	Set

Q 1. Find frequency for all numbers in an array. You are given Q queries, print freq for each query element.

ex ⇒ arr[10] = {2, 6, 3, 8, 2, 8, 2, 3, 8}

Q[4] = {2, 8, 3, 5}

↓ ↓ ↓ ↓

2 3 2 0

Brute force

for each query ⟶ Q

iterate entire array & find freq] → N

TC ⇒ O(Q * N)

SC ⇒ O(1)

Hashmap < key, value >

key → element of array → INT

value → freq of element → INT

// Hashmap <INT, INT> hm;

{ 2, 6, 3, 8, 2, 8, 2, 3, 8 }

Q => [2, 8, 3, 5]
↓ ↓ ↓ ↓
3 3 2 0.

hm

<2, 3>
<6, 1>
<3, 2>
<8, 3>

// pseudo

```
HashMap<INT, INT> hm;
for(i=0; i<N; i++){
    if( arr[i] is present in hm ){
        // update the value of arr[i], by +1
    }
    else
    { insert (arr[i], 1)
    }
}
```

→ search(arr[i])

→ update

TODO

O(N)

TC => O(N+Q)
SC => O(N)

```
for( i=0; i< Q.length; i++ ){
    x = Q[i]
    if( x is present in hm ){
        print( hm[x] )
        ↳ value of key x
    } else print(0)
}
```

O(Q)

Q2. Find the first non-repeating element

ex ⇒ arr[6] ⇒ [1 2 3 1 2 5]

✗ ✗ ✓ ✗ ✗ ✓

o/p = 3

arr[8] ⇒ [4 3 3 2 5 6 4 5]

✗ ✗ ✗ ✓ ✗ ✓ ✗ ✗

o/p = 2

arr[7] = [2 6 8 4 7 2 9]

✓

o/p = 6

Soln

1) create a hm

2) update hm with element, frequencies

~~[(3) iterate the hm, print first key that has value = 1.]~~

*** in hashmap order of insertion is not maintained.

<1,1>		<6,10>
<2,3>	→ no order in hashmap	<1,1>
<4,5>		<2,3>
<6,10>		<4,5>

iii) iterate array for each arr[i] and get freq. if freq == 1, return arr[i]

```
for (i=0; i<N; i++) {
        freq map → hashmap containing freq. of all elements
}

for (i=0; i<N; i++) {
    x = arr[i]
    freq = hm[x]  ← get value for key x
    if (freq == 1)
        return arr[i]
}
return -1
```

TC ⇒ $O(N+N) = O(2N) \approx O(N)$

SC ⇒ $O(N)$

Q 3. Given N arr elements, find no. of distinct elements.

ex ⇒ arr[5] ⇒ { 3, 5, 6, 5, 4 } ⇒ o/p = 4.

arr[7] ⇒ { 6, 3, 7, 3, 8, 6, 9 } o/p = 5.

1) Hashmap → freq map

<6, 2>
<3, 2>
<7, 1>
<8, 1>
<9, 1>

map.size() → O(1)

3, 5, 6, 5, 4

hashset

3
5
6
4

set.size()

** if you insert duplicate key in hashset, it would make no impact

```
// hashset<Integer> hs;

    for(i=0; i<N; i++){
        hs.insert(arr[i])
    }
return hs.size()
```

TC ⇒ O(N)
SC ⇒ O(N)

Q4. Given N arr elements, check if all elements are distinct or not.

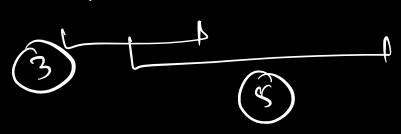
ex ⇒ {3, 4, 8, 3, 8} → true / false

{1, 6, 8, 2, 4} → true / false

```
for(i=0; i<N; i++){
    hs.insert(arr[i])
}
return hs.size() == N
```

Q Given a string calculate length of longest palindromic substring

ex ⇒ a b a c a b



o/p = 5

a b c d e

o/p = 1

Brute force : check for all possible substrings

$N^2 * N \Rightarrow O(N^3)$

ex ⇒

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
x	b	d	y	z	z	y	d	b	d	y	z	y	d	x

P_1 P_2 c

palindrome

length ⇒ $(P_1 \quad P_2)$

$= P_2 - P_1 - 1$

$= 14 - 8 - 1$

$= \underline{\underline{5}}$

* 1) take every character as centre, and expand on centre till they are equal ⇒ [max^m odd length palindromic substr]

```
int expand ( string s, p1, p2) {
        while ( p1 >=0 && p2 < N &&
                s[p1] == s[p2] ) {
            p1 --
            p2 ++
        } return p2 - p1 - 1
}
```

ex ⇒ a b c b a

P_1 P_2

ex ⇒ b d y z z y d b

P_1 P_2

pseudo

```
ans = 1
for( i = 0; i<N; i++) //odd length
    ans = max( ans, expand( str, i, i) )
}
for( i = 0; i<N-1; i++){ //even length
    ans = max( ans, expand(str, i, i+1))
}
return ans;
```

$$\boxed{\begin{array}{l} TC \Rightarrow O(N^2) \\ SC \Rightarrow O(1) \end{array}}$$