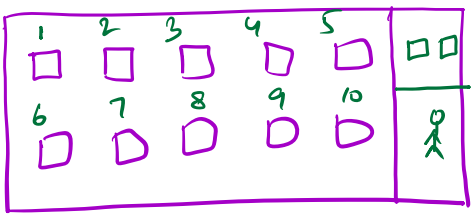
Our attitude towards life determines life's attitude towards us.

— John Mitchell

→ HashMap Intro
→ Frequency of each query
→ First non-repeating element
→ # distinct elements
→ Subarray with sum = 0

HashMap Intro

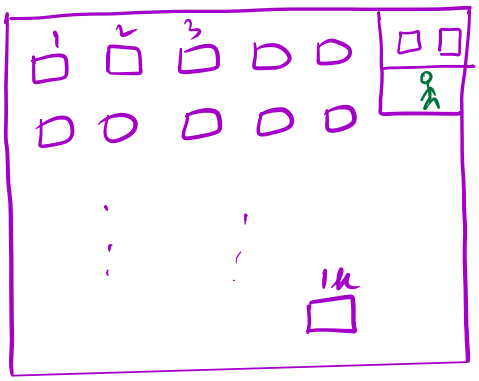
a)



→ Register

Room Nr	Availability
1	✗
2	✗
3	✓
4	✓

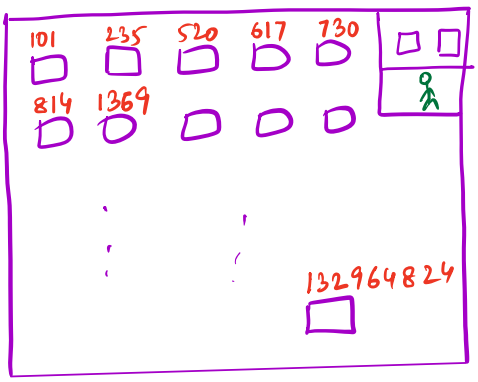
b)



Room Numbers → 1 to 1000

boolean room[1001];

c)



room no. $\in [1, 10^9]$

1000 room no.

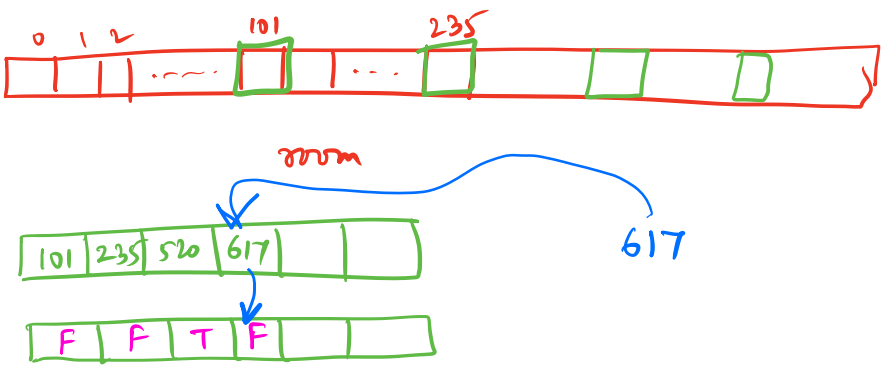
boolean room[10^9+1];

Huge space wastage.

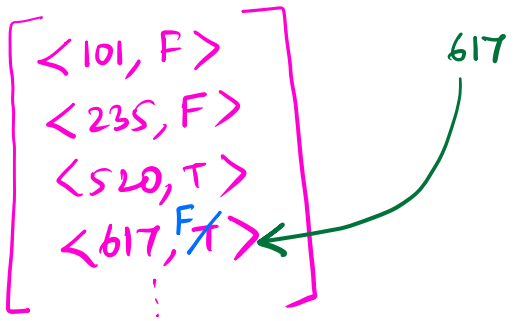
O(1) T.C.

room no 5213 :-

room[5213] == T/F.



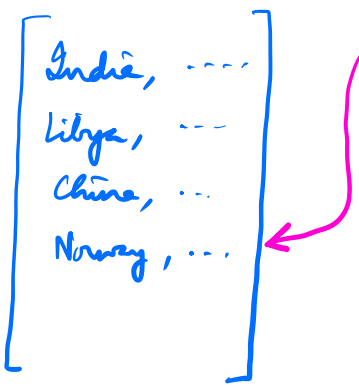
HashMap<key, value>



T.C. → O(1) to search/update
S.C. → O(n) to store n entries

keys are unique
values can be anything.

Q1) Store population of every country.



key → country name (String)
value → population (long)

<String, long>

HashMap<String, Long> hm;

Q2) No. of states in each country

HashMap<String, Integer> hm;

String → country name
Integer → No. of states

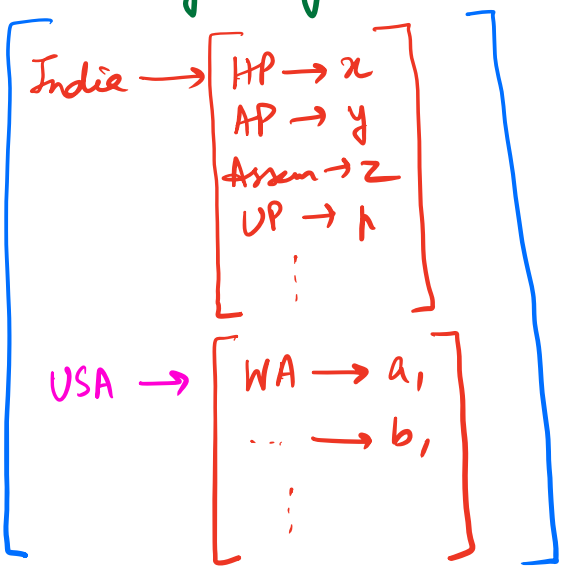
Q3) For every country, we want to know all state names

```
HashMap<String, ArrayList<String>> hm;
```

String → country name

ArrayList<String> → List of states

Q4) For every country, store population of all states.




```
HashMap<String, HashMap<String, Long>> hm;
```

country name → state name → population

population of all states.

Hash Map

<key, value>

insert → <key, value>
(put) hm.put(key, value);

search → key
hm.containsKey(key); → T/F

delete → key
hm.remove(key);

update → <key, value>
hm.replace(key, value)

get → key
hm.get(key); } provides the value.

size
hm.size();

~ getOrDefault

HashSet

<keys>

insert → <key>
(add) hs.add(key);

search → key
hs.contains(key);

delete → key
hs.remove(key);

size
hs.size();

Every operation is O(1) T.C

[Break till 10:42 PM]

hs → {3, 1, 5, 4, 16}

hs.contains(x) ⟶ T/F

Q1) Find the frequency of numbers.

Given ar[n], m queries, for each query find the frequency of the queried element.

$1 <= n <= 10^5$, $1 <= m <= 10^5$, $1 <= ar[i] <= 10^9$.

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

m = 4 {2, 8, 3, 5}

2 ⟶ 3

8 ⟶ 3

3 ⟶ 2

5 ⟶ 0

Idea 1:- Brute force.

No. of queries * T.C. of complete iteration of array

= m * n

= O(m*n) T.C.

Idea 2:- Using HashMap <int, int>

<2,3> <8,3>

<6,1> <10,1>

<3,2>

key → array element

value → frequency

```
void printFreq(int ar[], int q[]){
    int n = ar.length;
    int m = q.length;
    HashMap<Integer, Integer> hm = new HashMap<>();
    for(int i=0; i<n; i++){
        if(hm.containsKey(ar[i])){
            hm.put(ar[i], hm.get(ar[i])+1);
        }
        else{
            hm.put(ar[i], 1);
        }
    }
    for(int i=0; i<m; i++){
        if(hm.containsKey(q[i])){
            print(hm.get(q[i]));
        }
        else{
            print(0);
        }
    }
}
```

Q2) Find the first non-repeating element of an array.

ar[6] = {1 2 (3) 1 2 5} ans = 3

ar[8] = {4 3 3 (2) 5 6 4 5} ans = 2

ar[7] = {2 (6) 8 4 7 2 9} ans = 6.

Idea 1 :- Brute Force

$O(n^2)$ T.C., O(1) S.C.

Idea 2 :- Populate HashMap with <elem, freq>

Iterate on HashMap ⟶ Does not work because HashMap does not store elements in order.

```
for(int x : hm.keySet()){
    if(hm.get(x)==1){
    }
    else{
    }
}
```

Idea 3 :- Populate HashMap with <elem, freq>

Iterate on array and find first element for which hm.get(x)==1.

O(n) T.C.

O(n) S.C.

Q3) Given ar[n], find the no. of distinct elements.

ar[5] = {3 5 6 5 4} ans = 4

ar[5] = {1 1 2 1 2} ans = 2

ar[3] = {3 3 3} ans = 1

Insert all array elements in HashSet

No. of keys = size of HashSet ⇒ gives the ans.

```
HashSet<Integer> hs = new HashSet<>();
for(int i=0; i<n; i++){
    hs.add(ar[i]);
}
return hs.size();
```

Q4) Given ar[n], check if all the elements are distinct or not.

```
HashSet<Integer> hs = new HashSet<>();
for(int i=0; i<n; i++){
    hs.add(ar[i]);
}
return (hs.size()==n);
```

Q5) Given ar[n], check if there exists a subarray with sum = 0.

H/W