Hashmap Introduction

id	marks
57	96
2	95
9	82
11	85
47	56

Hashmap → key - value pairs

get marks
update marks
remove id
add a new id - marks pair

→ O(1)

57 → 96
2 → 95
9 → 82
11 → 85
47 → 56

HashMap

map

~~57 → 95~~

52 → ~~97~~ 98

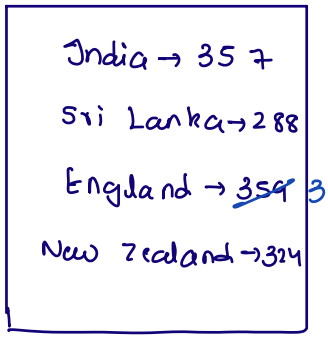
2 → 82

4 → 90

functions of hashmap :

```
void main() {

    HashMap<Integer, Integer> map = new HashMap<>();

    map.put(57, 95);
    map.put(52, 97);        → map.put(key, value)
    map.put(2, 82);
    map.put(4, 90);
    map.put(52, 98);

    sopln(map.get(57));     95
    sopln(map.get(50));     null

    sopln(map.containsKey(57));     true
    sopln(map.containsKey(45));     false

    map.remove(50);     (nothing will happen)
    map.remove(57);
    sopln(map.size());
```

put, get, containskey, remove, size → O(1)



map.get("India");

357 is scored by how many countries

put, get, containsKey, remove, size → O(1)

traverse HashMap → O(n)

n : map.size()

→ todo (covering in next class)

hint : getting all key of HM

Q.1 **Count frequency**

Given an array and Q queries, find how many times a particular element is coming in array.

A: 2 1 2 3 1 5 4 2 1

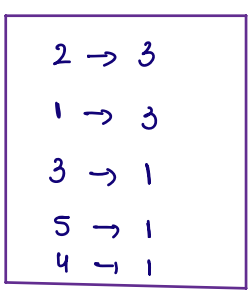
Queries

ele	ans
2	3
1	3
9	0
5	1

idea1 : for every query travel the entire array and get ans. TC: O(Q*N)

idea2 : create frequency map of A[].

A: 2 1 2 3 1 5 4 2 1

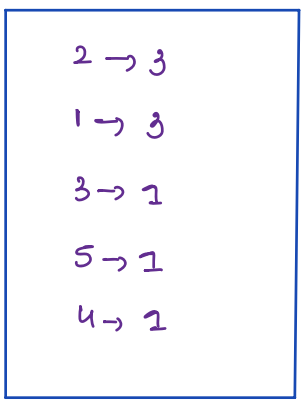


map

Key → ele

value → freq

A : 2 1 2 3 1 5 4 2 1



map

map.containsKey(A[i])

T

```
int temp = map.get(A[i]);
temp++;
map.put(A[i], temp);
```

F

```
map.put(A[i],1)
```

```
void solve (int [] A, int [] Q) {
    int n = A.length;
    // create freq. map
    HashMap<Integer, Integer> map = new HashMap<>();
    for (int i=0; i<n; i++) {                          --> N itr
        if (map.containsKey(A[i]) == false) {
            map.put(A[i], 1);
        }
        else {
            int temp = map.get(A[i]);
            temp++;
            map.put(A[i], temp);
        }
    }

    // go on every query and give answer
    for (int i=0; i<Q.length; i++) {                   --> Q itr
        int ele = Q[i];
        if (map.containsKey(ele) == false) {
            System.out.println(0);
        }
        else {
            System.out.println(map.get(ele));
        }
    }
}
```

TC: O(N+Q)

SC: O(N)

Q.2 Given an array A[], find first non repeating element.

A = 2 5 4 5 2 6 ans = 4

A = 4 5 4 4 3 4 ans = 5

A = 2 5 4 5 2 6
0 1 2 3 4 5

2 → 2
5 → 2
4 → 1
6 → 1

Idea: i) Create a freq map

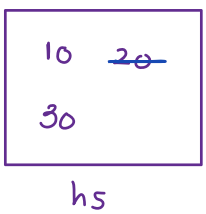
ii) Travel A[], and first ele with freq=1 is ans.

Note: order of insertion is not maintained in Hashmap.

Hashset Intro → It stores keys
↳ unique keys

```
void main() {
    HashSet<Integer> hs = new HashSet<>();
    hs.add(10);
    hs.add(20);
    hs.add(30);
    hs.add(10);
    SOPln(hs.contains(20));
    hs.remove(20);
    SOPln(hs.size());
}
```

hs.add(key) → (nothing)



true

2

add, contains, remove, size → O(1)

Q.3 Given an array A[], find total no. of distinct elements.

A = 3 9 3 4 5 ans : 4

A = 3 3 3 4 4 ans : 2

A = 3 9 3 4 5

1) idea1 : HashMap

→ create freq. map

ans: map.size();

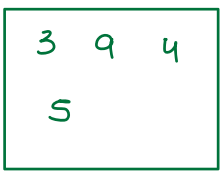
3 → 2
9 → 1
4 → 1
5 → 1

2) idea2 : HashSet

(freq. of ele is not our concern in this question)

A = 3 9 3 4 5

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



hs

Q.4 Given an A[], find if it has any subarray with sum = 0. {Google}

↳ continous part of array

	0	1	2	3	4	5	6	7
A:	2	4	-1	3	-2	5	1	6

ans: true

	0	1	2
A:	2	4	3

ans: false

ideal: Do on every subarray and find sum from s to e using prefix sum.

```
boolean solve(int[] A) {
    for(int s=0; s<n; s++) {
        for(int e=s; e<n; e++) {
            sum of subarray from s to e
            → using PS / CF
            if(sum==0) {
                return true;
            }
        }
    }
    return false;
}
```

itr: $\frac{N(N+1)}{2}$

TC: $O(N^2)$

Expected TC: O(N)

	0	1	2	3
A:	1	-3	2	4
PS	1	-2	0	4

→ sum of subarray 0,i

```
if (PS[i] == 0) {
    return true;
}
```

	0	1	2	3	4	5	6	7
A:	2	4	-1	3	-2	5	1	6
PS:	2	6	5	8	6	11	12	18

ans: true

PS[1] = PS[4]

sum(0,1) = sum(0,4)

~~sum(0,1)~~ = ~~sum(0,1)~~ + sum(2,4)

sum(2,4) = 0

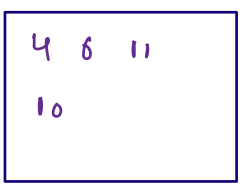
i) If ps[i] = 0, return true

ii) If value in ps[] is repeated, return true.

```
boolean solve (int [] A) {
    int [] ps = prefixSum (A);
    for (int i = 0; i < ps.length; i++) {
        if (ps[i] == 0) {
            return true;
        }
        hs.add (PS[i]);
    }
    if (hs.size() != ps.length) {
        return true;
    }
    else {
        return false;
    }
}
```

	0	1	2	3	4	5
A =	-4	1	5	-2	7	0
PS :	-4	-3	2	0	7	7

	0	1	2	3	4	5
A =	4	2	5	-1	-4	0
PS :	4	6	11	10	6	6



hs

TC: O(n)

SC: O(n)

Doubt

3 5 10

	max	min	diff
{ }	0	0	0
{3}	3	3	0
{5}	5	5	0
{10}	10	10	0
{3 5}	5	3	2
{3 10}	10	3	7
{5 10}	10	5	5
{3 5 10}	10	3	7
			21

	0	1	2
	3	5	10
	↓	↓	↓
A[i] is →	1	2	4

smax = 3 + 10 + 40 = 53

max in

how many subseq.

A[i] is min in $2^{(n-i-1)}$

	0	1	2
	3	5	10
A[i] is →	4	2	1

min in

how many subseq.

smin = 12 + 10 + 10

= 32

diff = smax - smin

= 53 - 32 = 21

```
Arrays.sort(A);

int smax = 0;

for(int i=0; i< A.length; i++) {
        // A[i] is max in 2^i subseq.
        smax += A[i] * (1<<i);
}

int smin = 0;
for(int i=0; i< A.length; i++) {
        // A[i] is min in 2^(n-i-1) subseq.
        smax += A[i] * ( 1 << (n-i-1));
}
```

(1<<i) → 2^i

(1 << (n-i-1)) → 2^{n-i-1}