Todays Content

- Hashmap Intro
- Frequency of each query
- first non repeating element
- Dist elements
- # Subarray with sum = 0

Given N Student ids & marks for each student & Q Queries

3 Types of Queries are present

Type1: Given id, check if id exist or not

Type2: Given id, return marks of that student

Note: if id not present return 0;

Type3: Given id, Inc marks of student by 10

Note: if id not present, no need to update

Note: Every Student will have a different id

Ex: 10 student

	0	1	2	3	4	5	6	7	8	9
Id[10] :	24	67	90	40	67	77	44	20	19	28
Mark[10] :	~~678~~ 688	894	400	364	700	800	400	900	650	800

Q = 6

Type	id :	ans :
1	40	True
1	65	False
2	44	400
2	19	650
3	24	update ✓
3	70	noupdate

Idea: For Q Queries TC = Q * N = QN

Type1: Iterate on id[] & check
if present: True else: False
TC: O(N)

Type2: Iterate on id[] & get index &
get marks at that index
if not present: 0
TC: O(N)

Type3: Iterate on id[] & get index &
Inc marks at that index by 10
if not present: nothing
TC: O(N)

Optimization:

1. Store data in hashmap ?
2. In hashmap each data entry is stored in <key, value> pair
3. Note: Keys are distinct in hashmap

	0	1	2	3	4	5	6	7	8	9
Key Id[10] :	24	67	90	40	67	77	44	20	19	28
Mark[10] :	678	894	400	364	700	800	400	900	650	800

3. Above data in hashmap

[<24 678> <40, 364> <67 894> <67, 700> <90 400> <77, 800> <44 400> <20, 900> <19 650> <28, 800>]

Q = 6

Type	id:	ans:
1	40	True
1	65	No
2	44	400
2	19	650
3	24	update
3	70	noupdate

Why hashmap?

1. Insert a <key, value> pair
2. To Search For a key
3. To Get Value For a key
4. To Update Value For a key
5. To Delete a <key, value> pair

Each of above operation takes O(1) ? Future

Idea2: With hashmap

Step1: Insert all student id, marks in hashmap.

TC: N * O(1) = O(N) SC: O(N)

Step2: For Q Queries

- Single Type1: O(1)
- Single Type2: O(1)
- Single Type3: O(1)

TC: Q * O(1) = O(Q)

Overall TC: O(N+Q) SC: O(N)

Syntax: `HashMap<KeyType, ValueType> hm = new HashMap<>();`

↳ hashmap name

int : Integer	float : Float	char : Character	Future
long : Long	double : Double	String : String	

Q1. Store population of every country in HashMap<Key, Value>

Key : CountryName → 1. unique

Value : Population 2. Retrival happens based on name

Syntax: `HashMap<String, Long> hm = new HashMap<>();`

Q2: Store frequency of all arr[] elements in arr[] eg: ar[5] = {3 1 3 6 1} (0 1 2 3 4)

hm { 3 : 2, 1 : 2, 6 : 1 }

Key: unique arr[] element

Value: freq of arr[] element

Syntax: `HashMap<Integer, Integer> hm = new HashMap<>();`

Most Used functions in HashMap<Key, Value>

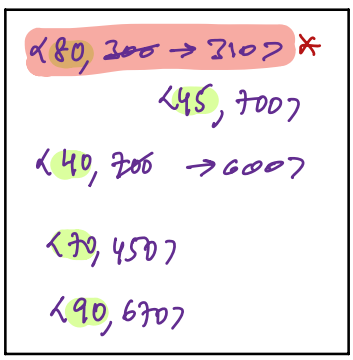
1. size() : return no: of keys present in HashMap
2. put(key, value) : Insert following key value pair in HashMap.
3. containsKey(key) : Return True/False based on current key present.
4. get(key) : If key present in hashmap Return value of given key.
   If key not present in hashmap Return null.
5. remove(key) : If key present. Remove pair, with given key,
   If key not present, don't do anything.
6. update() : We don't have any predefined function to update value of key

Q1: For Every Student Id & Marks are given store them

Syntax: `HashMap<Integer, Integer> hm = new HashMap<>();`

Insert: `hm.put(key, value);`

```
hm.put(80, 300); ✓
hm. put (40, 700); ✓
hm. put (90, 670); ✓
hm. put (45, 700); ✓
hm. put (70, 450); ✓
```



Search: `hm.containskey(key)`

```
if( hm.containskey (80)) {
    print("Student present);
}
else{
    print ("Absent");
}
```

Output: Student present

```
if( hm.containskey (95)) {
    print("Student present);
}
else{
    print ("Absent");
}
```

Output: Absent

Acess: `hm.get(key)`

```
print (hm.get(40))
```

output: 700

```
print (hm.get(95)
```

output: null // not there

Update: No function

```
hm.put(40, 600);
```

Q: Inc value of key 80 by 10?

1.
```
int cd = hm.get(80);
hm.put(80, cd+10)
```

2.
```
hm. put (80, hm.get(80)+10);
```

Delete

```
hm.remove(80)
print (hm.get(80))
```

null

Note in HashMap:

1. Keys are distinct in hashmap
2. Order: Is not maintained for keys? No Future
3. TC: If key is number/char/boolean: TC: O(1)? Future

   put(key, value) containsKey(key) get(key) remove(key):

   If key is String: TC: O(l) // l is length of String.? ⇒ Future

   put(key, value) containsKey(key) get(key) remove(key):

HashSet<Key>

1. Store only keys & They have to be unique

HashSet<Key_Type> hs = new HashSet<>();

int : Integer	float : Float	char: Character
long : Long	double: Double	String: String

Functions:

1. size() : return no: of keys present in hashset
2. add(key) : Insert following key in hashset
3. contains(key) : Return T/F based on current key
4. remove(key): if key present. Remove key
   if key not present, don't do anything.

Note:

1. Order: Is not maintained for keys.
2. TC: If key is number/char/boolean : TC: O(1) ?
   add(key) contains(key) remove(key):
   If key is String: TC: O(l) // l is length of String ? Future
   add(key) contains(key) remove(key):

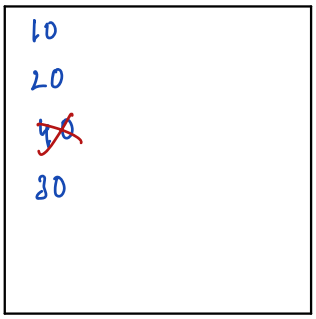
Create:

```
HashSet<Integer> hs = new HashSet<>();
```

add:

```
hs.add(10) ✓
hs.add(20) ✓
hs.add(40) ✓
hs.add(30) ✓
print(hs.size()) // 4
hs.add(40) // If key is already present, it won't add again
print(hs.size()) // 4
```

hashset hs



remove():

```
hs.remove(40) // * delete
hs.remove(80) // do nothing
```

contains():

```
if( hs.contains(30)) {
    print(present)
}
```

output: present

```
if( hs.contains(60)) {
    print(present)
}
```

output: no output

18) Find Frequency of Numbers

Given ar[N] elements & Q queries, for each query find frequency of each element in array.

Constraints:

$1 \le N \le 10^5$

$1 \le Q \le 10^5$

$1 \le ar[i] \le 10^9$

Ex: ar[11] =

0	1	2	3	4	5	6	7	8	9	10
2	6	3	8	2	8	2	3	8	10	6

Queries: 4

2 : 3

8 : 3

3 : 2

5 : 0

Ideal: For every query, iterate on arr[] & get count

TC: Q*N[] SC: O(1)

Idea2: Store elements in hashmap<key, value> hm

key = arr[] element value = it's frequency

<2,3>

<6,2>

<3,2>

<8,3>

<10,1>

Queries

2 : 3

8 : 3

3 : 2

5 : 0

TC: N*O(1) + Q*O(1) = N+Q SC: O(N)

```
void printFreq(int ar[], int qr[]){   TC: O(N+Q)   SC: O(N)
    HashMap<Integer, Integer> hm = new HashMap<>();
    int n = ar.length;
    // Insert all ar[] in hm
    for(int i=0; i<N; i++){
        // Search ar[i] in hm
        if(hm.containsKey(ar[i])){ // ar[i] present, update freq+1
            int f = hm.get(ar[i])
            hm.put(ar[i], f+1)
        }
        else{
            hm.put(ar[i], 1);
        }
    }
    // Get freq for all queries
    int Q = qr.length;
    for(int i=0; i<Q; i++){
        // get freq qr[i]
        if(hm.containsKey(qr[i])){
            print(hm.get(qr[i]));
        }
        else{
            print(0);
        }
    }
}
```

Q8: Given ar[N] ele, find no: of distinct elements

Constraints: Every ele consider 1 time

$1 \le N \le 10^5$

$1 \le ar[i] \le 10^9$

ar[5] = 3 5 6 ~~5~~ 4 ans = 4

ar[5] = 1 ~~2~~ ~~2~~ 2 ~~2~~ ans = 2

ar[3] = 3 ~~3~~ ~~3~~ ans = 1

ar[7] = 6 3 7 ~~8~~ 8 ~~8~~ 9 ans = 5

Ideas: Add all elements in hashset & get size

6 3 7 (3) 8 (6) 9 → hashset: 6 3 7 8 9 = 5

int distinct(int ar[]) { TC: N * O(1) = O(N) SC: O(N)

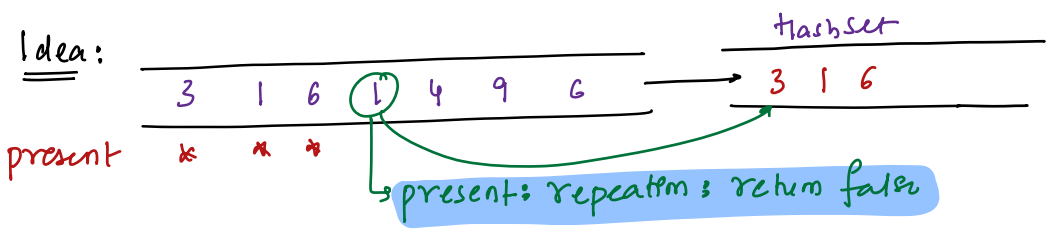
```
int n = ar.length;
HashSet<Integer> hs = new HashSet<>();
for(int i=0; i<n; i++){
    hs.add(ar[i]);
}
return hs.size();
}
```

38) Given ar[N], check if all elements are distinct or not?

ar[5] = 6 8 3 2 7 : Yes

ar[7] = 3 1 6 1 4 9 6 : No

Idea:




```
boolean checkDistinct(int ar[]) {   TC: O(N)  SC: O(N)
    int n = ar.length;
    HashSet<Integer> hs = new HashSet<>();
    for(int i=0; i<n; i++){   ---> N * { O(1) + O(1) } = O(N)
        // search ar[i] in hs
        if( hs.contains(ar[i])) {
            return false
        }
        else
            hs.add(ar[i])
        }
    }
    return true;
}
```

48) Given ar[N] elements, check if there exists a **Subarray with Sum = 0**

(continous part of arr[])

Constraints:

$1 \le N \le 10^5$

$1 \le ar[i] \le 10^9$

Sum of all ele: min : 1 max : 10^{14}

	0	1	2	3	4	5	6	7	8	9	
ar[10] :	2	2	1	-3	4	3	1	-2	-3	2	: return True

	0	1	2	3	4	
ar[5] :	2	4	-1	-2	1	: return false

	0	1	2	3	4	
ar[5] :	2	-1	0	2	3	: return true

Ideas: Generate all subarray sums, & check with 0

TC : O(N²) using carry forward

SC : O(1) no extra space

Idea1: Get pfsum[] for given arr[]

pfsum[i] = sum of all elements from 0..i

	0	1	2	3	4	5	6	7	8	9
ar[10] :	2	2	1	-3	4	3	1	-2	-3	2
psum[10] :	2	4	5	2	6	9	10	8	5	7

obs: Repeation in Psum[]?

Psum[8] = 5 Psum[2] = 5

Psum[8] = sum of [0..8]

Psum[8] = sum of [0..2] + sum of [3..8]

Psum[8] = Psum[2] + sum of [3..8]

5 = 5 + sum of [3..8]

sum of [3..8] = 0

Psum[3] = 2, Psum[0] = 2

Psum[3] = sum of [0..3] =

Psum[3] = sum of [0..0] + sum of [1..3]

Psum[3] = Psum[0] + sum of [1..3]

2 = 2 + sum of [1..3]

sum of [1..3] = 0

If there is a repeation in Psum[] ⇒ Subarray with Sum = 0

If psum[] contains = 0, Subarray with Sum = 0

Edgecase:

	0	1	2	3
ar[] =	2	-5	3	6
psum[] =	2	-3	0	6

: No repeation, But subarray with sum = 0 exists

↳ Psum[2] = 0 = Sum of [0..2] = 0 Subarray.

Edgecase: Way2 : Initialize hashset with 0?

We start summation with '0' hence insert at inside

	0	1	2	3
ar[ ] =	2	-5	3	6
psum[ ] =	2	-3	0	

hashSet : 0 2 -3 0 : repeatition

```
boolean SubZero(int ar[]) {   TC: O(N)   SC: O(N+N)
    int n = ar.length;
    HashSet<Integer> hs = new HashSet<>();
    hs.add(0); // Edgecase
    int pfsum[N];
    construct pfsum -> TODO
    for(i=0; i<N; i++){
        // Psum(i) is already coming
        if(hs.contains(Psum[i])){
            return true //Subarray with sum 0 exist
        }
        else
        {
            hs.add(Psum[i])
        }
    }
    return false;
}
```