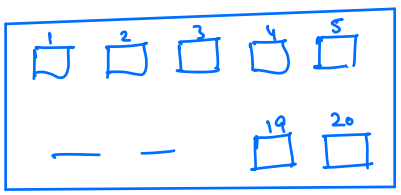
Today's Quote →

Today's Content

- → Hash Map Intro
- → Frequency of each query.
- → First non-repeating element
- → Distinct elements
- → Sub array with sum = 0

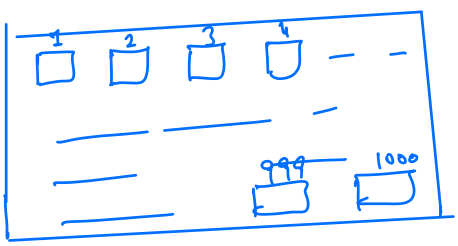
HashMap Intro

Parvan + Prerna



Room No.	Ava.
1	X
2	X
3	✓
4	X
5	✓
:	
20	X

funding.



[572].

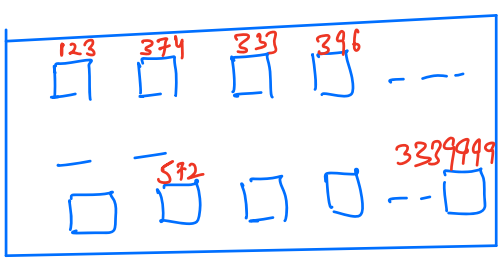
Room no's → [1 1000]

boolean [1001] arr

arr[5] → false.
not available

arr[572] = true.
Room available.

lucky no's.



$[1, 10^9]$

check if room no. 2375 is available or not?

→ boolean $[10^9+1]$ arr

Issue → Huge space wastage.

Adv. → TC : O(1)
for checking if any room no. is available or not.

We need to have a Key, value pair.

< Key, value >

[< 123 , ava. >
< 333 , not ava. >
< 3339999 , ava. >
< 572 , not ava. >
< 3279 , ava. >]

Check for room no. 572.
⇒ 572 is not available.

[T.C → O(1) for search.
S.C → O(N) for N rooms]

{ How? → Adv. module.
Implementation of HashMap. }

Q1) Store population of every country :

Key : country_name → String

Value : population → int / long

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

Q2) Number of states in every country :

Key : country_name → String

Value : no. of states → int

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

Q3) for every country we want to store all state names.

Key : country_name → String

Value : All state names → list of strings.

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

Q4) for every country, store population of every state.

Key : country_name

Value : Hashmap of string vs. integer.

```
HashMap < String, Hashmap <String, Integer>>
hm;
```

observation → Key must be unique

→ Value can be anything { int/long/list/hashmap/arr[] }

→ Key can be only primitive data type.

⇓

{ integer / long / float / double / char / string }

→ Hashset / set.

Hashmap functionality

<Key, value>

size() → count of unique keys present in the hashmap.

insert(key, value)

search(key)

delete(key)

update(key, value)

A single operation will take O(1) time.

Hashset functionality

<Key>

size() → count of keys in the hashset.

insert(key)

search(key)

delete(key)

update(key) ✗

Hashmap

<India, 131>
<China, 144>
<USA, 60>

insert(India, 160)

override value

<India, 160>

When you insert already existing key, then value of that key will be overriden.

Note → A single operation will take O(1) time.

→ If we are insert N <key, value> pairs.

T.C → O(N)

S.C → O(N)

Hashing library name in these languages →

pseudocode	Java	C++	Python	JS	C#
HashMap	HashMap	unordered_map<>	Dictionary	Map	Dictionary
HashSet	HashSet	unordered_set<>	set	set	HashSet

Q: **Find frequency of Numbers**

Given N array elements & Q queries. For each query, find frequency of each element in array.

constraints: $1 <= N <= 10^5$, $1 <= Q <= 10^5$, $1 <= arr[i] <= 10^9$

arr[11] : { 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
5 : 0

idea-1.

For every query, iterate on array & get count.

T.C → O(Q·N), S.C → O(1)

Idea 2. {store data in HashMap}

Key → array element. → integer
value → frequency of element ↔ integer.

hm < integer, integer >

arr[11] : { 2 6 3 8 2 8 2 3 8 10 6 }
(0) (1) (2) (3) (4) (5) (6) (7) (8) (9) (10)

element		frequency
2	→	~~1 2~~ 3
6	→	~~1 2~~
3	→	~~1 2~~
8	→	~~1 2~~ 3
10	→	1

pseudo-code

```
void printFreq ( int[] arr , int[] queries) {
    N = arr.length  // N → no. of elements
    Q = queries.length  // Q → no. of queries.
    HashMap < integer , integer > hm ;  // Exact syntax [TO DO]

    for( i = 0 ; i < N ; i++) {
        if ( hm.search (arr[i]) == true){
            // arr[i] is already present
            hm[arr[i]] += 1
        }
        else { // arr[i] is coming for first time
            hm.insert( arr[i], 1) ;
        }
    }                                              N

    for ( i = 0 ; i < Q ; i++){
        if ( hm.search(queries[i]) == true){
            print hm[queries[i]];
                  ↳ accessing the value corresponding to queries[i]
        }
        else{
            print 0 ;
        }
    }                                              Q
}
```

T.C → O(N+Q)
S.C → O(N)

[Break 10:03 → 10:10]

Q) Find the first non-repeating element. ⇒ { first element that is not repeating }

arr[6] : { 1 2 3 1 2 5 } ans : 3

arr[8] : { 4 3 3 2 5 6 4 5 } ans : 2

arr[7] : { 2 6 8 4 7 2 9 } ans : 6.

Idea-1: Store the elements in Hash map.
Iterate on the hashmap & get our ans. ✗

arr[6] : { 1 2 3 1 2 5 }

element	freq.
5	1
1	~~1~~ 2
2	~~1~~ ~~2~~ 2
3	1

Note → Order of insertion of elements is not maintained in the Hashmap.

Idea-2. Store the elements in Hash map.
Iterate on the array & get our ans.

T.C → O(N)
S.C → O(N)

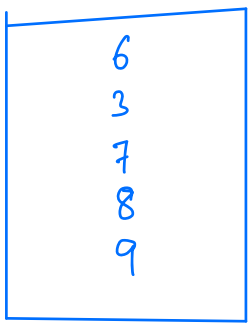
{code → ToDo}

Q) Given arr[N] elements, find no. of distinct elements.

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

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

arr[7] = { 6 3 7 3 8 6 9 } ans = 5



[In hashset, if you insert any key multiple times, it will contain only one occurence.]

pseudo-code:

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

T.C → O(N)
S.C → O(N)

Q) Given arr[N] elements. Check if all elements are distinct or not?

arr[5] : { 6 8 3 2 7 } : true

arr[7] : { 3 1 6 1 4 9 6 } : false

idea: insert all the elements in hashset.

```
if ( size of array == size of hashset) {
        return true
}
else {
        return false
}
```

T.C → O(N)

S.C → O(N)

Q: Given arr[N] elements. Check if there exists a subarray with sum = 0. [true / false]

arr[10]: { 2 2 1 -3 4 3 1 -2 -3 2 } : true.
0 1 2 3 4 5 6 7 8 9

idea: Consider all subarrays.
for every sub-array, calculate sum. [if sum == 0] return true

- 3 nested loops: T.C → $O(N^3)$, S.C → $O(1)$
- Prefix-sum: T.C → $O(N^2)$, S.C → $O(N)$
- Carry forward: T.C → $O(N^2)$, S.C → $O(1)$

arr[10]: { 2 2 1 -3 4 3 1 -2 -3 2 }
0 1 2 3 4 5 6 7 8 9

psum[10]: { 2 4 5 2 6 9 10 8 5 7 }

observation: Some numbers in psum[] are repeating. ⇒ [There exists a subarray with sum = 0]

① psum[2] = 5 = sum(0-2)
psum[8] = 5 = sum(0-8) = sum(0-2) + sum(3-8)
5 = 5 + sum(3-8)
sum(3-8) = 0

② psum[0] = 2 = sum[0-0]
psum[3] = 2 = sum[0-3] = sum[0-0] + sum[1-3]
2 = 2 + sum[1-3]
sum[1-3] = 0

Doubt:

arr[4]: { 2 (0) -5 (1) 3 (2) 6 (3) }

pSum[4]: { 2 -3 0 6 }

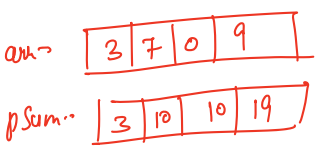
in pSum() no repeation, but we have a subarray with sum = 10

pSum[2] = sum[0-2] = 0

Find observation

① → if elements are repeating in pSum[] } There exists a subarray with sum = 0

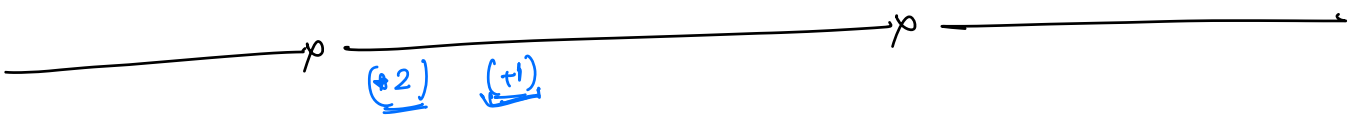
② → if 0 is present in psum[] }



pseudo-code.

```
boolean checkSubarraySum(arr, N){
    //Create pSum[]
    HashSet<int> hs;
    for(i=0; i<N; i++){
        if(psum[i]==0){return true}
        hs.insert(psum[i])
    }
    if(hs.size()<N){return true}
    else {return false}
}
```

T.C → O(N)
S.C → O(N)



Alex. → Ø 1.

[A] ← target.

(240) = (1 1 1 1 0 0 0 0)

ans- count of set-bits.

```
                1 0        (1<<1) = 1x2.
                1 1        (take help) +1
              1 1 0        (<<1) *2
              1 1 1        (take help) +1
            1 1 1 0        (<<1) *2
            1 1 1 1        (take help +1)
        1 1 1 1 0 0 0 0    (<<4)
```