# Hashing - 1

## Content

→ Hashmap intro

→ freq. of each element

→ first non-repeating element

→ # of distinct elements

→ exist a subarray with sum=0

Scenerio-1 : 1000 rooms labelled as: [1, 1000]

↳ occupied / not occupied

array ⇒ bool room[1001]

since rooms are labelled as [1,1000] not [0, 999]

⇒ room[i] = true [if ith room is occupied]

⇒ room[i] = false [not occupied]

Scenerio 2 : 1000 rooms labelled between $[1, 10^9]$

bool room$[10^9+1]$

↳ Issue : Huge space wastage

↳ Advantage : TC: O(1) to find any room's occupancy

## Hashmap

It stores <key, value> pairs

<10015, occupied>
<123, unoccupied>
⋮
N entries
= 1000

⇒ check in 10015?
→ occupied in TC: O(1)

TC: O(1) to search

SC: O(N) to store for N rooms.

Note: Keys are unique. Value can be anything.

Q. Store population of every country.

key: country name → string
value: population → int/long

$\overset{\text{key}}{\text{string}}$, $\overset{\text{value}}{\text{long}}$

Hashmap<string, long> hm; ⇒ pseudo syntax

↑ variable name

eg India, US, UK

hm = <"India", $1.5 \times 10^9$>
<"US", $10^8$>
<"UK", $10^7$>

Q2 for every country, we want to know all states.

Key : country name → String

Value : all states name → array<string>

↳ C++ : vector
↳ py : list
↳ java : arraylist

Hashmap<string, array<string> > hm

Q3 for every country, store population of each state.

Key : country name → string

Value : population of each state } → Hashmap<string, long>

$$\underset{\text{state name}}{\uparrow} \qquad \underset{\text{population}}{\uparrow}$$

Hashmap<string, Hashmap<string, long >> hm

Observation 1 : value can be anything

Observation 2 : Key can only be primitive datatype

↙

int / long / float / double / string / char

Hashset <key>

→ we only store keys
→ keys have to be unique
→ only primitive datatype

## Hashmap functions

size : {# of keys}

insert (key, value)

search (key) → value
↳ NOT FOUND

delete (key) → delete key & value

update (key, newValue)

↳ Hashmap

~~<India, 800>~~

<US, 200>

<India, 900>

## Hashset functions

size : {# of keys}

insert (key)

search (key) → true
↳ false

delete (key)

$\boxed{\text{All operations above are } O(1)}$

→ Hashing libraries name in diff. languages.

| Pseudocode | Java | Cpp | Python | JS | C# |
|---|---|---|---|---|---|
| Hashmap | Hashmap | unordered_map | dict | map | dictionary |
| Hashset | Hashset | unordered_set | set | set | Hashset |

## Question 1

Given N array elements & Q queries.
for each query, find freq. of given element in array.

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

| Q=4 | freq |
|---|---|
| 2 | 3 |
| 8 | 3 |
| 3 | 2 |
| 5 | 0 |

Constraints:

$1 <= N <= 10^5$ $1 <= Q <= 10^5$

$1 <= a[i] <= 10^9$

Idea 1: for each query, iterate & get count

TC: $O(Q*N)$ SC: $O(1)$

Can't create a count array since a[i] <= $10^9$.

Idea 2: Store data in hashmap

key → array element → int

value → freq. of element → int

{ 2 6 3 8 2 8 2 3 8 10 6 }

| | |
|---|---|
| <2,3> | <8,3> |
| <6,2> | <10,1> |
| <3,2> | |

Code

```
Hashmap <int,int> hm

for (i=0; i<n; ++i) {        → O(N)
    //key = a[i]
    if (hm.search(a[i]) == true ) {
            hm[a[i]]++   //update
    }
    else {
        hm.insert( a[i], 1 ) //insert
                         ↑
                        freq
    }
}
```

```
for (i=0; i<Q.size ; ++i) {        → O(Q)
    // Key = Q[i]
    if (hm.search(Q[i]) == true) {
        print( hm[Q[i]])    → access value of key
    }
    else {
        print(0)
    }
}
```

TC : O(N+Q)

SC : O(N)

Question 2: find the first non-repeating element.

↳ first element from start

eg a[6] = { 1 2 3 1 2 5 }     ans = 3

freq=2   freq=2   freq=1

a[ ] = { 4 3 2 2 5 4 }

Idea 1 :
1. Insert all elements in hashmap
2. Iterate hashmap to get first key with value=1.

Note : Order of insertion of keys is not maintained in hashmap/ hashset.

Idea 2 : 1. Insert all elements in hashmap $\rightarrow O(N)$
2. Iterate over array & get first element with hm[a[i]] = 1 $\rightarrow O(N)$

TC : O(N)   SC : O(N)   $\boxed{\text{Code} \rightarrow \text{TODO}}$

Question 3 :

Given a[N] elements, find no. ~~of~~ $\overset{\text{count}}{}$ distinct elements ?

eg  a[5] = { 3 5 6 5 4 }   ans = 4

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

Idea 1 : 1. insert all elements in hashmap
2. count keys with value = 1

Idea2 : Insert all elements in hashset

a[7] = { 6 3 7 3 8 6 9 }

Hashset<int> hs

hs : { 6 3 7 8 9 }   $\boxed{\text{hs.size} = 5}$

Note: In hashset, if same key is inserted multiple times, it will only store 1 occurance.

Code

```
Hashset<int> hs
for (i=0; i<n; ++i) {
    hs.insert(a[i])
}
print( hs.size)
```

TC: O(N)

SC: O(N)

Question 4

Given N elements, check if all elements are unique or not ?

1. insert all elements in hashset

2. check if hs.size() == n ⇒ unique

else

Not unique

## Question 5

Given a[N] elements, check if there exists a subarray with sum = 0.

eg a[10] =

|  |  |  |  |  |  |  |  |  |  |
|---|---|---|---|---|---|---|---|---|---|
| 2 | 2 | 1 | -3 | 4 | 3 | 1 | -2 | -3 | 2 |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |

2+1+3=0

-3+4+2+1+-2+-3 = 0

ans = true

Idea: for every subarray, calculate sum

nested loops $O(N^3)$

prefix sum $O(N^2)$, $O(N)$

carry forward $O(N^2)$, $O(1)$

TC: $O(N^2)$ SC: $O(1)$

|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| a[10] = | 2 | 2 | 1 | -3 | 4 | 3 | 1 | -2 | -3 | 2 |
| pf[] = | 2 | 4 | 5 | 2 | 6 | 9 | 10 | 8 | 5 | 7 |

observation: In pf[], numbers are repeating pf[0] = 2 = sum[0,0]

pf[3] = 2 = sum[0,3] = sum[0,0] + sum[1,3]

$\cancel{2}$ = $\cancel{2}$ + sum[1,3]

$\boxed{\text{sum}[1,3] = 0}$

a[4] = { 2 -5 3 6 }

pf[] = { 2 -3 0 6 }

$\hookrightarrow$ in pf[], there is no repetition but subarray with sum = 0 exist?

pf[2] = 0 = sum[0,2]

Note: In pf[], even if single 0 is present, there exists a subarray with sum = 0

final Idea:

If ele repeat in pf[]
OR
If 0 is present in pf[]

$\rightarrow$ there exist a subarray with sum = 0

Code

```
bool subarrayZero(a[]) {
    n = a.length
    pf[n]    // create pf[]    → TODO
    Hashset<int> hs
    for (i=0; i<n; ++i) {
        if (pf[i]==0) { return true }
        hs.insert(pf[i])
    }
    if ( hs.size() < N) {    // repetition in pf[]
        return true
    }
    return false
}
```

TC: O(N)

SC: O(N)