Hashing - 1

Content

→ Hashmap intro

→ freq. of each element

→ first non-repeating element

→ # of distinct elements

→ exist a subarray with sum=0

Scenario 1 : 1000 rooms labelled as : [1, 1000]

↳ occupied / not occupied

array ⇒ bool room[1001]

since rooms are labelled from [1,1000] NOT [0, 999]

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

⇒ room[i] = false [not occupied]

Scenario 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

⇒ 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

Q1 Store population of every country

Key: country name → String
value: population → int / long

HashMap<String, Long> hm ⇒ pseudo syntax
(String = Key, Long = value, hm = variable name.)

eg: India, US, UK

hm = <"India", 1.5×10^9>
<"US", 10^8>
<"UK", 10^7>

Q2 for every country we want to know all states.

Key: country name → string

value: all state names → 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>

state name

population

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 datatypes

Hashmap functionality

size : {# of keys present}

insert (key, value)

search (key) → value
↳ NOT FOUND

delete (key)

update (key, newValue)

↳ Hashmap

~~<India, 800>~~

<US, 200>

<India, 900>

Hashset functionality

size : {# of keys present}

insert (key)

search (key) → true / false

delete (key)

All operations here are O(1)

→ Hashing libraries name in diff. languages

Pseudocode	Java	C++	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 arr[10] = { 2 6 3 8 2 8 2 3 8 10 6 }

Q =	freq
2 :	3
8 :	3
3 :	2
5 :	0

Constraints:

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

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

Idea 1: for every query, iterate & get count

TC: O(QN) SC: O(1)

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

Code

```
HashMap<int,int> hm    // TODO

for(i=0; i<n; ++i) {                    → TC : O(N)
    // Key = a[i]                           SC : O(N)
    if( hm.search( a[i]) == true ) {
        hm[a[i]]++  // Update
    }
    else {
        hm.insert( a[i], 1 )   //insert
    }
}

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

TC : O(N+M)
SC : O(N)

Question 2

find the first non-repeating element.

↳ first element from start, non-repeating

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

freq=2 freq=2 freq=1

a[7] = { 4 3 3 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 → O(N)
2. Iterate over array & get first element with hm[a[i]] = 1 → O(N)

TC : O(N) SC : O(N)

Code → TODO

Question 3

Given a[N] elements, find no. of (count) distinct elements?

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

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

Idea

Insert all elements in Hashset

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

Hashset<int> hs

hs = { 6 3 7 8 9 }

hs.size = 5

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

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

eg a[10] =

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

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]

2 = 2 + sum[1,3]

sum[1,3] = 0

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

pf[] = { 2 -3 0 6 }

↳ in pf[] there is no repetition but subarray with sum=0 exist ?

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

Note: In pf[] every 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[]
→ there exist a subarray with sum=0

Code

```
bool subarrayZero(a[]) {
    n = a.length
    pf[n]  // construct 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)