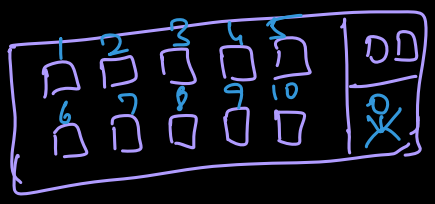
# Today's Content

- → Hashmap Intro
- → Freq. of each char
- → First non-repeating char
- → # distinct elements
- → # subarr sum=0

## Hashmap Intro :

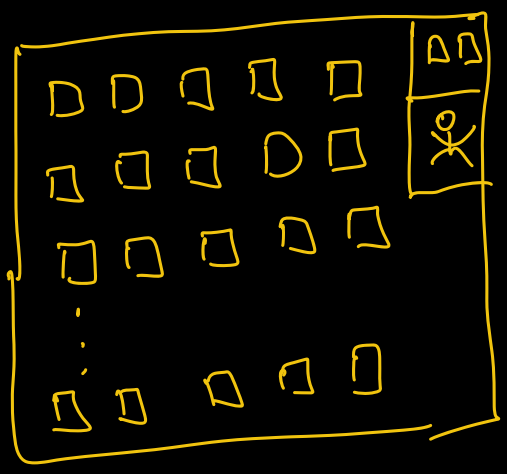
(a) Ravalika + Ashish



Register

| Room No | Occupied |
|---|---|
| 1 | ✗ |
| 2 | ✗ |
| 3 | ✓ |
| 4 | ✓ |

(b)



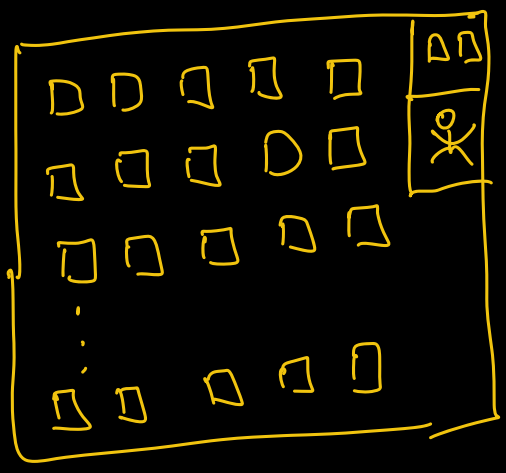
Rooms : 1 – 1000

bool room [1001]

room [1]
room [3] →
room [7]

TC : O(1) → to check if a room is avail or not

(C) Covid



1000 lucky numbers

Range $[1 - 10^9]$

bool rooms $[10^9 + 1]$

1000 numbers in

* Huge space is wasted ✗
* Adv: Access is O(1) ✓

Retain the advantage & discard the disadvantage
= Hashmap

Hashmap

<79, occupied>
<85, occupied>
<93, occupied>
<113, occupied>
<65, occupied>

85 ✗ not available
65 ✓ available

Key : Value pair

TC: O(1)
SC: O(N)
↓
no. of entries in the hashmap

∴ Search is on Room No.

Note:
∴ Keys are unique
∴ Values can be anything

Qn: Store population of every country in Hashmap

Key: Country name → String

Value: population count → long

`Hashmap<String, long> hm`

Qn: No. of states in each country

Key: Country name → String

value: no. of states → int

`Hashmap<String, int> hm`

Qn: For every country, we want to know **all state names**

Key: Country name → String

value: single state name* : List<String>

- C++: vector
- Java: ArrayList
- Python: List

`Hashmap<String, List<String>> hm`

Qn: For every country, store population count of each state.

Key: Country name → String

Value: population of each state → Hashmap<Key, Value>

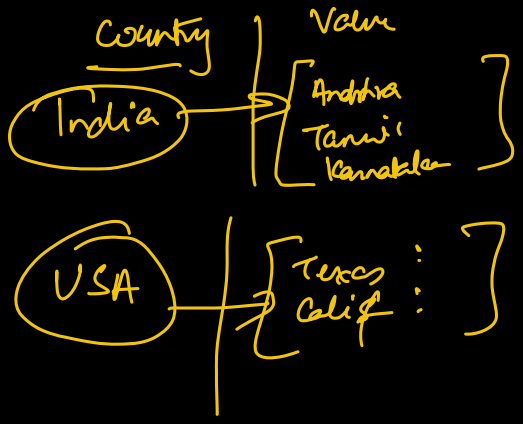
Key → State name (String)

Value → population Count (long)

`Hashmap<String, Hashmap<String, long>> hm`

String → Key

Hashmap<String, long> → Value




- Hash Set

HashSet <Key>

## Hashmap functionality v/s HashSet functionality

<Key, Value>

size : { No. of keys }

get (key)

insert (key, value)

search (key)

delete (key)

update (key, value)

O(1) time

↳ <India, 800> → 810 (over written)

<USA, SD>

Update India . 810

* When you insert same key, value will be overwritten.

<Key>

size : { No. of keys }

insert (key)

search (key)

delete (key)

O(1) time

~~update (key)~~

↳ [<India>, <Bharat>]

Note: A single operation takes O(1) time

if we insert N <key, val> pairs

TC : O(N)

SC : O(N)

# Hashing Library name in diff. language

| Pseudocode | Java | C++ | Python | JS | C# |
|---|---|---|---|---|---|
| Hashmap | Hashmap | unordered_map | dictionary | map | Dictionary |
| Hashset | Hashset | unordered_set | set | set | HashSet |

## Q1: Find Frequency of Numbers

Given arr[N], Q queries. For every query find the frequency of element in array.

arr[] = { 2 6 3 8 2 8 2 3 8 10 6 }

Q = 4 freq

2 : 3
8 : 3
3 : 2
5 : 0

Constraints

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

# idea 1: For every query, iterate the array & get frequency.

TC: O(Q×N)
SC: O(1)

idea 2: Store data in hashmap
key : array element
value : freq of that element
Hashmap<int, int> hm

Pseudo Code:

```
void printFreq(int arr[], int Q[]) {
    int n = arr.length
    int m = Q.length
    Hashmap<int, int> hm
    for(i=0; i<n; i++) {
        if(hm.search(arr[i]) == true) {
            // arr[i] is already present
            hm[arr[i]]++
        }
        else {
            hm.insert(arr[i], 1);
        }
    }
    for(i=0; i<m; i++) { // iterate over queries
        if(hm.search(Q[i]) == true) {
            print(hm[Q[i]])
        }
        else {
            print(0)
        }
    }
}
```

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

O(N)

Q

2 : 3
3 : 2
8 : 3
5 : 0

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

Hashmap:

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

Qn: Find first non-repeating element (from start)

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 2 6 8 4 7 2 9 } ans = 6

idea1: Insert all elements in Hashmap & iterate on hashmap & get 1st key with val = 1 ✗

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

Hashmap

[ <1,2> <5,1> <2,2> <3,1> ]

idea2: Insert all elements in Hashmap & iterate on array & get 1st key with val = 1 ✓

Step 1: Insert all elements in the hashmap with their frequency.

Step 2: Iterate on the array[] & get first element with freq = 1

TC: O(N)

SC: O(N)

ToDo

Break: till 8.45 am

Qn: Given arr[N], find the no. of distinct elements

↳ Consider each element once.

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

(3, 5, 6, 4)

ans = 4

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

(1, 2)

ans = 2

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

(6, 3, 7, 8, 9)

ans = 5

idea: insert all elements in a hashset.

Hashset <int> hs

6, 3, 7, 8, 9

hs.size()

Pseudocode:

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

TC: O(N)

SC: O(N)

Qn: Given arr[N]. Check if all elements are distinct.

arr[5] = { 6 8 3 2 7 } ✓ return true

arr[7] = { 3 1 6 1 4 9 8 } ✗ return false

idea: insert all elements into hashset.

if(hs.size < N) { return false }

else { return true }

TC: O(N)

SC: O(N)

Qn: Given arr[N]. Check if there exists a subarr (continuous part of array) with sum = 0

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

↳ ans = true

[1-3] : [2 1 -3] = 0

[3-8] : [-3 4 3 1 -2 -3] = 0

ideas: For every subarray, calc sum == 0

↓ Nested loops

O(N^3)

↓ for every subarr Pf sum

O(N^2)

↘ Using carry fwd

O(N^2)

TC: O(N^2)

SC: O(1)

idea 2: Create pf array.

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

Obs1: Some no. are repeated.

pf[2] = 5 sum[0 2]

pf[8] = 5 sum[0 8]

sum[0 8] = sum[0 2] + sum[3 8]

5 = 5 + 0

sum[3 8] = 0

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

pf[3] = 2 = sum[0 3]

sum[0 3] = sum[0 0] + sum[1 3]

2 = 2 + 0

∴ In pf[], if numbers are repeating

∴ There exists a subarray with sum = 0

Doubt:

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

pf[4] = { 2 -3 0 6 }

subarr = 0

[0 2]

∴ if in pf[], there is a zero

↳ There is a subarr sum = 0

**Final obs:**

If element repeats in pf[] array
If 0 is present in pf[] array
] There exists a subarr with sum = 0

**Pseudo code:**

```
bool subarrZero(int a[]) {
    int n = a.length
    int pf[n]
    // Construct pf[n]
    Hashset<int> hs
    for(i=0; i<n; i++) {
        if(pf[i]==0) { return true }
        hs.insert(pf[i])
    }
    if(hs.size < n) { return true }
    else { return false }
}
```

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

**Doubts:**

pf[3 8] = pf[8] - pf[2]

pf[8] = pf[2] + pf[3 8]

[0 8] [0 2] + [3 8]