Today's Quote →

Today's content

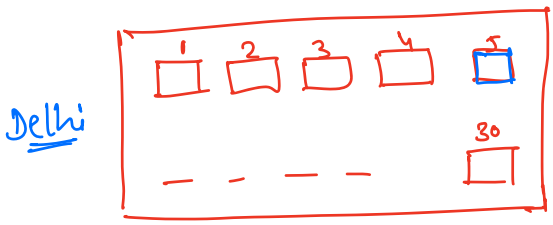
- → Hash Map Intro
- → Frequency of each query.
- → First non-repeating element
- → Distinct elements
- → Sub array with sum = 0

Break

[Dussehra : 4th - 5th October
Diwali : 22nd - 26th October]

HashMap Intro

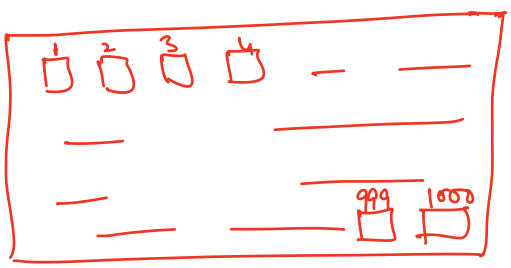
① Aravindhan + Meenakshi



→ Register

Room No.	Available
1	✗
2	✗
3	✗
4	✗
5	✓
.	
.	
30	✓

② Aravindhan + Meenakshi



[1000's] ⇒ [1 - 1000]

→ boolean arr[1001]

⇓

index → 0 to 1000

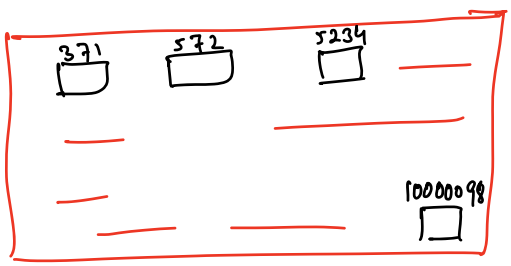
Check for room = 3

arr[3] : false.

Chuck for room = 572

arr[572] = true.

③ Aravindhan + Meenakshi



1000 no's in range [1, 10^9]

→ boolean arr[10^9+1]

Disadvantage → Huge space wastage.

Advantage → TC : O(1) [arr[i]

[If you need to check, room availability].

<Key, Value>

<79, ava.>

<95, ava.>

<572, ava.>

check for room = 85

If Key is not there, room 85 is not available.

check for room = 572

[572 is available.]

T.C → O(1) for Key search in hashmap.

S.C → O(N)

[Note:- → Keys must be unique.

→ Value can be anything.]

Q1) Store population of every country :

Key : country name → String

Value : population → int / long

HashMap < String , Long > hm .

Q2) Number of states in each country :

Key : country name → String

Value : # no. of states → int

HashMap < String, int > hm

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

Key : country name → String

Value : All states names → List < String >

↳ dynamic arrays.

HashMap < String, List< String> > hm

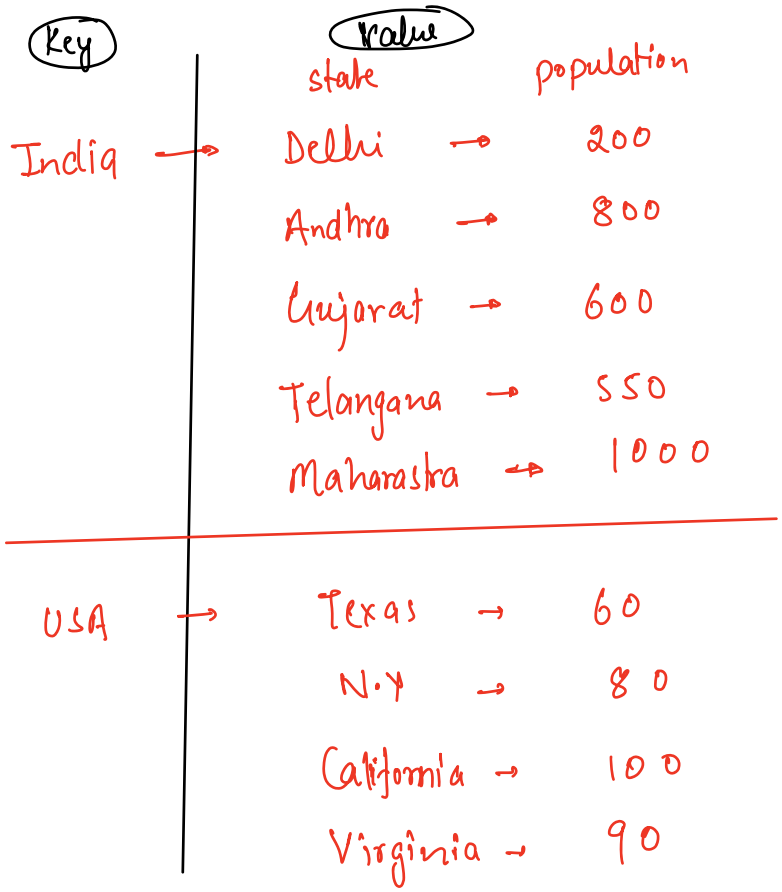
Q4) for every country , store population of each state.

Key : country name → String

Value : population of every state.] ⇒ HashMap < Key, value >

↳ state name ↳ population.

HashMap < String , HashMap < String , Long >> hm.



observation

① → value can be anything.

② → Key can be of primitive data-types only.

integer | long | float | double | String | character.

HashSet:

→ we only store keys.

→ keys have to be unique.

Hashmap functionality

< Key, value >

size → {Number of keys in a hashmap}

insert (Key, value)

search (Key)

delete (Key)

update (Key, value)

↳ India → ~~143~~ 170

China → 165

U.S.A → 80

again insert. (~~India → 170~~)

When we insert the same Key, it will override the previous value.

A single operation will take O(1)

Hashset functionality

< Key >

size → {no. of keys in the hashset}

insert (Key)

search (Key)

delete (Key)

update (Key) *

[Hashset
→ India *
→ Bharat]

[implementation of Hashmap
⇓
Adv. module.]

Conclusion → A single operation in hashmap / hashset ⇒ O(1)

↳ If we insert N < Key, Value > pairs.

T.C → O(N)

S.C → O(N)

Hashing library name in Java & python

pseudo-code	Java	Python
HashMap	HashMap	Dictionary.
HashSet	HashSet.	Set.

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 = ?	freq.
2 :	3
8 :	3
3 :	2
5 :	0

idea:
For every query, iterate & get count.
{T.C → O(N*Q), S.C → O(1)}

idea2: store data in hashmap.
Key → array element → (integer)
value → frequency → (integer)
HashMap < integer, Integer > hm.

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

Key.	Value.
2	3
6	2
3	2
8	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 < int , int > hm ;
    for ( i = 0 ; i < N ; i++) {
        if ( hm.search( arr[i] ) == true) {
                // arr[i] is already present
            hm [ arr[i] ] += 1   // increment the freq by 1
        } else {
            hm.insert ( arr[i] , 1 );
        }
    }                                          → N

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

T.C → O(N+Q)

S.C → O(N)

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.

ideas:

~~①~~ Insert all the elements in hashmap.

iterate in hashmap and find the first element with freq = 1

arr → { 1 2 3 2 4 1 }
0 1 2 3 4 5

Key	value
4	1
1	2
2	2
3	1

Note → order of insertion of key is not maintained.

✓② Insert all the elements in hashmap.

iterate in array and find the first element with freq = 1

arr → { 1 2 3 2 4 1 }
0 1 2 3 4 5

Key	value
4	1
1	2
2	2
3	1

⇒ [first non-repeating element]

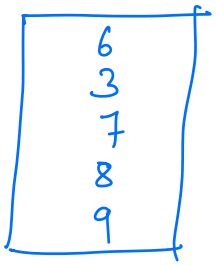
T.C → O(N)
S.C → O(N)

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

arr[5] = { 3 5 6 5 4 } ans = 4. [Consider every unique element.]

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

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



ans = hs.size()

Note:→ In hashset, if you re-insert a key, nothing will happen. [only 1 occurence will be stored].

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 elements in hashset.

```
if (hs.size() == N){
    // all elements are distinct
    return true
}
else { // elements are repeating
    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 }
0 1 2 3 4 5 6 7 8 9

Idea 1

① for every subarray, calculate sum & check if sum == 0

T.C → O(N²)
S.C → O(1)

- 3 nested loops → O(N³)
- prefix sum → O(N²)
- carry forward. → O(N²)

idea 2.

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

pSum: [2 4 5 2 6 9 10 8 5 7]

observation: In pSum[], no's are repeating.

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)

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: [2 (0) -3 (1) 0 (2) 6 (3)]

⇒ In pSum[] no repeating elements but we do have a sub-array with sum = 0.

pSum[2] = 0

sum[0-2] = 0

[Note:→ In your pSum[], if a single zero is present, then we have a sub-array with sum = 0]

Final observation

① if elements are repeating in pSum[]

② if 0 is present in pSum[]

] ⇒ subarray with sum = 0

arr→ | 1 | 2 | 0 | 4 |

pSum→ 1 3 3 4

arr→ | 0 | 1 | 2 | -3 |

pSum→ | 0 | 1 | 3 | 0 |

pseudo-code.

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

T.C → O(N)
S.C → O(N)

Doubts:

Min. value of $A \wedge X + B \wedge X$.

1^1 = 0
0^0 = 0
0^1 = 1
1^0 = 1

A → 0 1 0 1 1 0 1 0
^ 0 0 0 1 1 0 0 0
0 1 0 0 0 0 1 0
⇒ $2^6 + 2^1$

+

B → 0 0 1 1 1 0 0 1
^ 0 0 0 1 1 0 0 0
0 0 1 0 0 0 0 1
$2^5 + 2^0$

idea → unset the bit in both the numbers.

ans = $2^6 + 2^1 + 2^5 + 2^0$

A → 7 6 5 4 3 2 1 0
0 1 0 1 1 0 1 0
^ 0 1 1 1 1 0 1 1
0 0 1 0 0 0 0 1

B → 7 6 5 4 3 2 1 0
0 0 1 1 1 0 0 1
^ 0 1 1 1 1 0 1 1
0 1 0 0 0 0 1 0

⇒ $2^5 + 2^1$ + $2^6 + 2^1$

⇒ $(2^6 + 2^5 + 2^1 + 2^0)$

final observation → if ith-bit in both the no's.
→ 1, then in X it should be 1.
→ 0, then in X it should be 0

→ if ith-bit is different in both the no's.
it can be 0 or 1 in X.
for simplicity, consider it 0.

→ To achieve this, take AND of A, B.
[X = A & B]

→ $A \wedge (A \& B) + B \wedge (A \& B)$

T.C → O(1)
S.C → O(1)

X = 8

A = 6

B = 12.

```
A → 0 1 1 0        B → 1 1 0 0
X → 0 1 0 0        X → 0 1 0 0
    -------            -------
    0 0 1 0            1 0 0 0
```

X = 4

$2^1 + 2^3 = 10$ → minimum possible value.