Welcome 

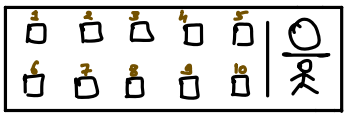
Agenda : Hashing

1) Hashmap Intro
2) frequency of each query
3) first non-repeating elements
4) # distinct elements
5) # subarray with sum = 0

Hashmap Intro

1)

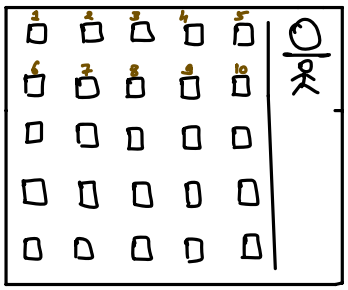
Aman & Susanta



Register

Room no	availability
1	✓
2	✓
3	X
4	X

1000 rooms.



T.C → O(1)

1 → 1000

array Reg [1001]

check in room 44 → constant time

↓

false

check in room 444

↓

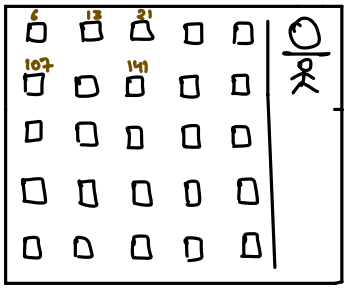
true

Siddhartha.

1000 lucky room no(s)

range: [1, 10^9]

↳ index / room no.s



Issue: space wastage

advantage: T.C → O(1)

Hashmap

Key Value pair

room no. room availability

<13, T>

<107, F>

<21, T>

Implementation of hashmap ⇒ Advanced

Check 107 ?

T.C: O(1) to search

S.C: O(N) for N rooms.

Note ① Keys are unique

② Values can be duplicate

Q1 Store population of every country

Key → Country → string

Value → Population → int / long.

hashmap < string , int/long > hm ← Representation of hash map.

(string → datatype Key; int/long → datatype Value.)

Q2 No. of states in each country.

Key → country → string

Value → #states → int

hashmap < string , int > hm

Q3 For every country we want to know all state names.

Key : Country ⟹ String

Value : list < states > ⟹ list < strings > ← Dynamic array

Java → vector

C++

hashmap < string , list < string > > hm

Q4 for every country store population of each state

Key : country name → string.

Value : population of each state → hashmap < string , int >

(string → State Name; int → State Popln.)

hashmap < string , hashmap < string , int > > hm

Takeaways

① Value can be anything

HashSet < Key >

→ When we only want to store Keys

→ Keys have to be unique.

Hashmap functionality < Key, Value >

① Search → (Key) O(1)

② Delete → (Key) O(1)

③ Insert → < Key, Value > O(1)

④ Update → < Key, Value > O(1)

Hashset functionality. < Key >

① Search → (Key) O(1)

② Delete → (Key) O(1)

③ Insert → (Key) → O(1)

④ Update → (Key) ✗

Hashmap
< India , 1000 >
< USA , 600 >
< India , 1400 >

overriding

⇒ When we insert same key again, it will override previous value.

T.A

→ A single operation in Hashmap / Hashset ⇒ O(1)

→ If we insert N < Key, value pairs > ⇒ O(N)

T.C → O(N) S.C → O(N)

Hashing library names in different languages

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

Constraints:

$1 \le N \le 10^5$ $\quad 1 \le Q \le 10^5$ $\quad 1 \le arr[i] \le 10^9$

eg:

arr: [2 6 3 8 2 8 2 3 8 10 6]

Q: 4

2 → 3

8 → 3

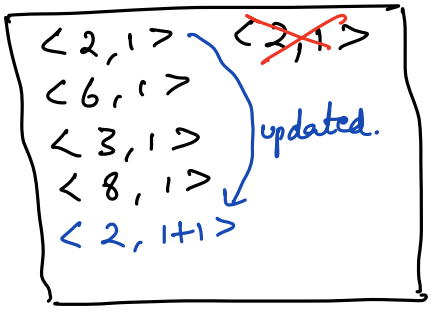
3 → 2

5 → 0

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

Idea 2: Store data in hashmap
Key → array elements → int
Value → freq. of element → int

hashmap<int, int> hm



→ When you are inserting data in hashmap, first check if key already exist. If it is present, then update Else insert Key Value pair.

Pseudo code

```
void PrintFreq ( int arr[] , int Q[] )
{
    int n = arr.length();
    int m = Q.length();
    hashmap < int , int > hm ;
    // Iterate array and store freq. of all elements.
    for( int i=0 ; i<n ; i++)
    {
        if ( hm.search (arr[i]) == true)
        {
            // update freq.
            hm[arr[i]] += 1 ;
        }
        else { // insert
            hm.insert ( arr[i] , 1)
        }
    }

    // Iterate all queries
    for( int i=0 ; i<m ; i++)
    {
        if ( hm.search ( Q[i]) == true)
        {
            // get value of key → Q[i]
            print ( hm[Q[i]] );
                    ↳ access value of key Q[i]
        }
        else {
            print (0) ;
        }
    }
```

T.C → O(N+Q)

S.C → O(N)

Q2 Find the first non-repeating element → first element from start, non-repeating.

eg:

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

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

Idea 1 → Insert all the elements in hashmap & iterate the hashmap to get 1st key with freq 1

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

hashmap :

<1,2>
<5,1>
<2,2>
<3,1>

Takeaway :
Order of insertion of keys is not maintained

Idea 2 : Insert all elements in hashmap & iterate array and get first elem with freq 1

Code

Step 1

① Insert element in hashmap. → O(N)

② Iterate array, get 1st element with freq 1 → O(N)

T.C → O(N) S.C → O(N)

H.W task

10:56

Q3 Given arr of size N, find no. of distinct elements.

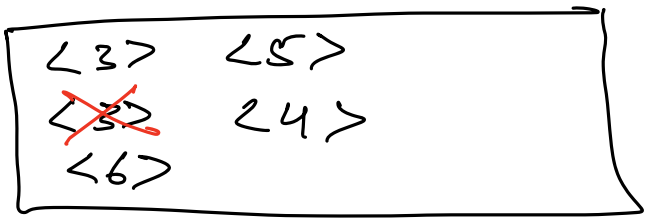
arr [5] : { 3 3 6 5 4 } ans = 4

arr [3] : { 3 3 3 } ans = 1

Ideas: insert elements in hashset

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

hashset



Q4 Given an array, check if all elements are distinct or not.

eg: arr[5] : { 6 8 3 2 7 } ✓ True

arr[7] : { 3 1 6 1 4 9 6 } ✗ False

Idea: ① Insert array elements in hashset

② if hashset.size() == arr.size()
return true.
else return false.

Pseudocode

H.W

Q5 Given an array of size N, check if there exists a subarray with sum == 0

eg: arr[10]

[2 2 2 1 -3 4 3 1 -2 -3 2]

Idea 1

For every subarray, calculate sum == 0 → T.C $O(N^2)$, S.C $O(1)$

- $O(N^3)$ — 3 nested loops
- $O(N^2)$ — Prefix Sum. S.C → $O(N)$
- $O(N^2)$ — carry forward. S.C → $O(1)$

Idea 2

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

obs 1

PF[2] = 5

PF[8] = 5

PF[8] = PF[0→2] + PF[3→8]

5 = 5 + PF[3→8]

PF[3→8] = 0

obs 1 ⇒ If PF[] numbers are repeating, ⇓ There exists a subarray with sum = 0

eg: arr[4] : { 2 -5 3 6 }

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

⇒ PF[2]

If in PF[] no repeatation, but subarray with sum = 0 present ⇒ PF[2]

obs 2 If in your PF[], even if single zero is present, there exists a subarray with sum = 0

Pseudocode

```
bool subarraySum( int arr[] )
{
    int n = arr.size();
    int pf[] ;  // TODO -> create prefix sum array.
    hashset<int> hs;

    for( int i=0; i<n; i++)
    {
        if( PF[i] == 0)
            return True;
        hs.insert( PF[i] );
    }
    if( hs.size() < n )
        return True
    else
        return False;
}
```

T.C → O(N)

S.C → O(N)