Lecture :- Hashing 1

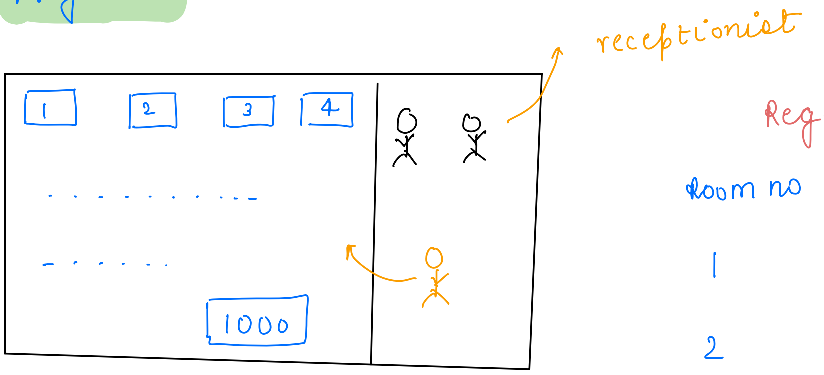
Agenda

- Introduction.
- frequency of each element
- first non-repeating element
- Hashset Introduction.
- # distinct elements.

Class starts at 7:05 AM

Introduction

Priyanka



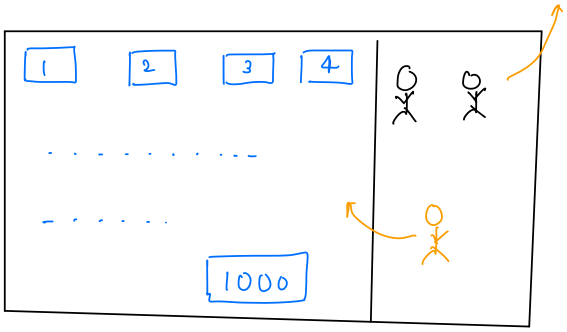
Register

Room no	Availability
1	✓
2	X
3	✓
4	✓
5	X
⋮	
1000	✓

Room ↑

searching through register

becomes tedious



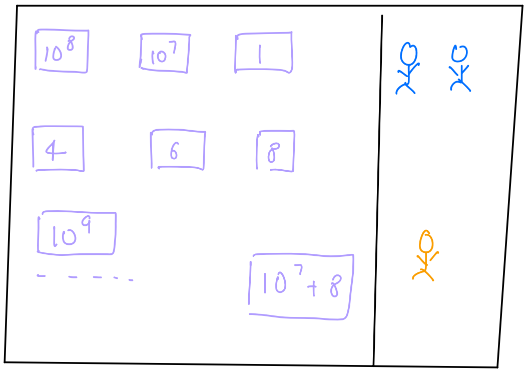
```
boolean[] availablerooms =
    new boolean[1001]
```

f	t	t	f	t
0	1	2	- - - - - - - - - -	10000

is room no 7068 empty?

availableRooms[7068] = true [empty]

= false [not empty]



boolean availableRooms[$10^9 + 1$]

t	f	t	t	t		f
0	1	2	- - - . . .			10^9

Issue: for 10,000 rooms, priyanka created $10^9 + 1$ size array

key [Room no]	Table	value [Availalibility]
1	—	true
2	—	false
3	—	false
4	—	true
5	—	true
⋮		⋮

saving lot of space ←

7068 is available? O(1) time complexity

Hashmap

table like structure [dynamic in size]

(key , value)

Things to know

```
Hashmap < key , value > map = new Hashmap<>();
```

Anything

Anything

- Integer
- Boolean
- String
- Advanced — ArrayList<>()
- Long
- Advanced — Student

Exampel: Store populatin of each country

key: country name → string

value: population → Long | Integer

`Hashmap<string, Integer> map = new HashMap<>();`

key	value
USA	330000..-
India	143000---
China	140000----

Ex2: for every country, we want to know state names

key: Country name (string)

value: All states List<string>

HashMap<string, ArrayList<string>> statesMap.

Ex3. for every country, store population of each state

key: Country name string

value: population of each state.

Map<string, Integer>

HashMap<string, HashMap<string, Integer>> map.

Operations on hashmap

1. **Insert**

HashMap<Integer, Boolean> map

map. put (2 , true)

map. put (3 , false)

map. put (4 , true)

map. but (2 , false)

(Integer) key	Boolean Value
2	~~true~~ false
3	false
4	true
2	false

All the keys of hashmap are unique

2. Size of hashmap => map.size() = 3

3 **Get value of a key:**

map.get (4) || true

map.get (2) || false

map.get(11) || null

4. Check whether a key is present or not?

map.containskey(4) || true

map. containskey(11) || false

All these operations have O(1) TC

Advanced Module

Qu Given arr[n] and Q queries, for every query, find freq of each element in array.

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

queries

2	2
8	3
10	1
49	0

```
void printfreq( int[] arr, int[] queries) {

Q  — for( int i=0; i< queries.length; i++) {

            int el = queries[i];
            int count =0;
n  —        for( int j : arr) {
                if (j == el) {
                    count++;
                }
            }
            print (count);
     }
```

TC: O(Q * n)

SC: O(1)

Approach2

key : array el (Integer)

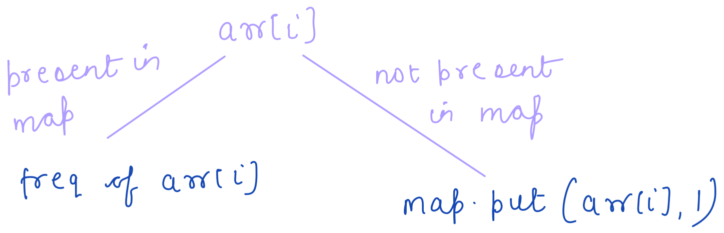
Value : count of each array el (Integer)

HashMap<Integer, Integer> freqMap.

0	1	2	3	4	5	6	7	8	9
2	6	3	8	2	8	3	8	10	1

arr[10] = [2 6 3 8 2 8 3 8 10 1]

Key	Value
2	~~1~~ 2
6	1
3	~~1~~ 2
8	~~1~~ 2
10	1
1	1



1. Get freq of arr[i]

2. freq = freq + 1

3. map.put(arr[i], freq)

map.put (arr[i], 1)

```
        void  printfreq( int[] arr, int[] Q ) {

            HashMap<Integer, Integer> map =

                        new HashMap<>();

o(n)  ——   for( int el : arr) {

      o(1) — if( map.containskey(el)) {
        o(1)  — 1.  int freq = map.get(el);
                2   freq = freq + 1;
        o(1)  — 3.  map.put( el, freq);

               } else {
        o(1)  ——    map.put(el, 1);

               }
          }

o(Q)  ——   for( int currEl : Q) {
      o(1) — if( map.containskey(currEl)) {
        o(1)  — print( map.get(currEl));

             } else {
                print(0);
             }
          }
      }
                        TC: o(n+Q)

                        SC: o(n)
```

Qu2: find first non- repeating el in array.

arr[6] = [1 2 3 1 2 5] ans = 3

arr[8] = [4 3 3 2 5 6 4 5] ans = 2

map:

6	-	1
4	-	2
3	-	2
5	-	2
2	-	1

Hash map does not maintains order of a key

↑

Advanced Module

```
int  firstNonRepeatingEl (int[] arr) {
          HashMap<Integer, Integer> map =
                        new HashMap<>();

o(n)  —  for( int el : arr) {
       o(1) — if( map.containsKey(el)) {
          o(1) — 1. int freq = map.get(el);
                 2  freq = freq + 1;
         o(1) — 3. map.put( el, freq);

               } else {
         o(1) —     map.put(el, 1);
               }
        }

        for (int el : arr) {
             if ( map.get(el) == 1) {
                  return el;
             }
        }

     return -1;
}
```

Break: 8:44 AM

Hashset

- hashmap but only key part
- when you don't need values

Things to know

Hashset< key > set = new Hashset<T();

↑ anything

operations

1. add — set. add(el).
2. size : set. size()
3. get : Get the value of a key { Invalid }
4. contains : set. contains (key)

Qu Given arr[n], find no of distinct elements

arr[5] = [3 5 6 5 4] ans = 4

arr[3] = [3 3 3] ans = 1.

hashset

arr[] = [3 5 6 5 4]

set = { 3 5 6 4 }

set.size() = ans

hashmap

key	value
3	1
5	2
6	1
4	1

map.size() = ans

```
int countDistinctIntegers( int[] arr){

    Hashset<Integer> set = new Hashset<>();

O(n) —  for( int el : arr){

    O(1) — set.add(el);
        }

O(1) — return set.size();
}
```

TC: O(n)

SC: O(n)

Google.

Qu Given arr[n], check if there exists a subarray with sum = 0

arr[10] = [2 2 1 -3 4 3 1 -2 -3 2]

Brute-force: Go to every subarray — check if sum is 0 or not?

TC: $O(n^2)$

SC: $O(1)$.

Expected TC: $O(n)$.

Approach: sum(l, r) ÷ prefix sum.

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

— $O(n)$

Observation

1. pf[i] = 0 (true)

└ There is a subarray from 0th idx to i th idx with sum = 0

$$\text{arr}() = \left[\begin{array}{cccccccccc} 0&1&2&3&4&5&6&7&8&9\\ 2&2&1&-3&4&3&1&-2&-3&2 \end{array}\right]$$

$$\text{pf}[] = [\ 2 \quad 4 \quad 5 \quad 2 \quad 6 \quad 9 \quad 10 \quad 8 \quad 5 \quad 7\ ] \text{ — O(n)}$$

Obs2:

pf[2] = 5

pf[8] = 5

— true (subarray with sum==0)

$\cancel{arr[0]} + \cancel{arr[1]} + \cancel{arr[2]} = \cancel{5}$ — I

$\cancel{arr[0]} + \cancel{arr[1]} + \cancel{arr[2]} + \quad = \cancel{5}$ — II

$arr[3] + arr[4] + arr[5] +$

$arr[6] + arr[7] + arr[8]$

I — II

$\{\cancel{arr[0]} + \cancel{arr[1]} + \cancel{arr[2]}\} - \{\cancel{arr[0]} + \cancel{arr[1]} + \cancel{arr[2]}$

$arr[3] + \ldots\ arr[8]\} = 5 - 5$

$arr[3] \ldots\ldots arr[8] = 0$

$arr[3] + arr[4] + arr[5] \ldots arr[8] = 0$

sumArray[3, 8] = 0

Challenge: Does pf[] contains repeating el or not?

```
boolean subArraySumwithZeroValue(int[] arr) {

    int[] pf = prefixoptimal(arr);
                       └ refer prefix sum class.

    Hashset<Integer> set = new Hashset<>();

    for( int el : pf) {
        if( el == 0) {
            return true;
        }
        set.add(el);
    }

    // Repeating el in array
    if( set.size() != arr.length) {
        return true;
    }

    return false;
}
```

TC: O(n)

SC: O(n)

Que: Given arr[n], check if all el are distinct or not?

arr[5] = [6 8 3 2 7] → true

arr[7] = [3 1 6 1 4 2 9] – false

hashset

arr = [6 8 3 2 7]

set = [6 8 3 2 7]

set.size == arr.length

- → all distinct
- → all are non-repeating

arr[] = [1 2 2 3]

set = [1 2 3]

set.size != arr.length

- → repeating el in array

hashmap

arr[5] = [6 8 3 2 7]

key	value
6	1
8	1
3	1
2	1
7	1

Thankyou ☺