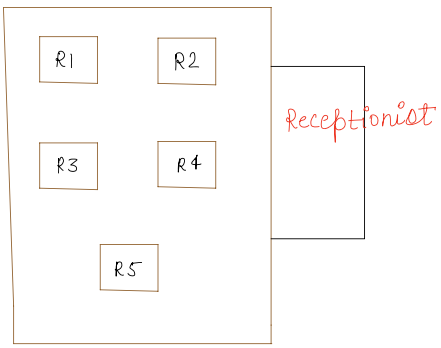
Lecture :- Hashing - 1

Agenda

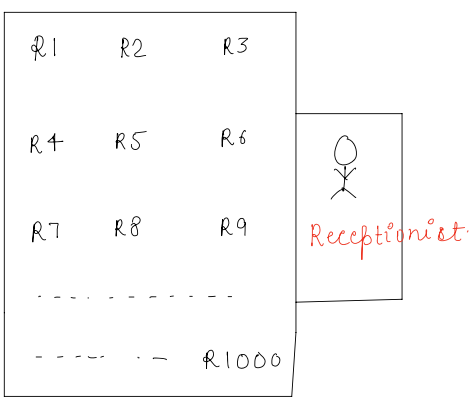
- Hashmap introduction
- Hashset introduction
- frequency of each element
- first non-repeating element
- count distinct elements
- subarray with sum 0

Hashmap



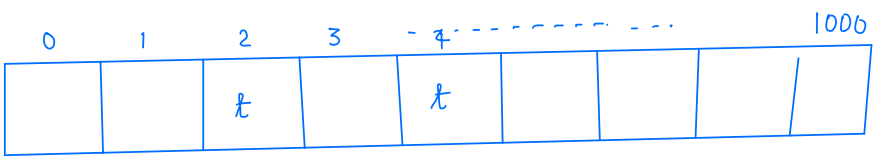
Solution : Registers

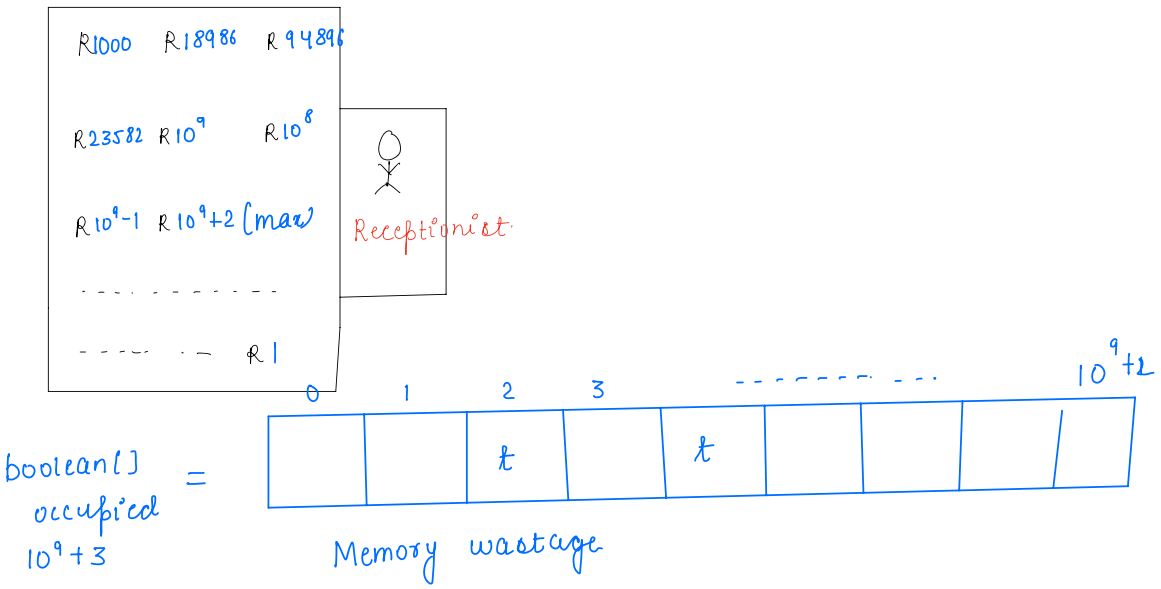
Room no.	Occupied
1	Yes
2	No
3	Yes
4	Yes
5	No

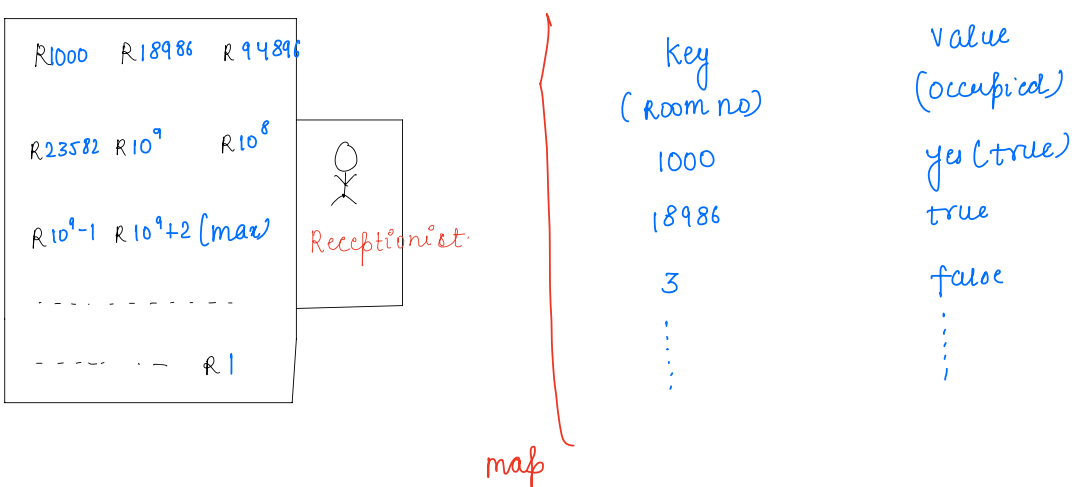


Room no	Occupied
1	Yes
2	No
3	Yes
4	No
⋮	
1000	Occupied

boolean[] occupied 1001 =







→ Map<key, value>
searching el
key → roomno (Integer)
value → Boolean.

→ key should always be unique.
→ value can be anything.
→ T.C of search = O(1).
→ key are unordered.

Quiz1 which of the following hashmap will you use to store the population of every country?

Key: String → country name

value: population → long

Map<String, Long> populationByCountry.

Quiz 2 which of the following hashmap will you use to store the number of states of every country?

Key: country name

value: Integer → no of states

Map<String, Integer>

Quiz 3 which of the following hashmap will you use to store the name of all states of every country ?

Key : Country name → String

Value: Name of all states → List<String>

Quiz 4 which of the following hashmap will you use to store the population of each state in every country ?

Key : Country name

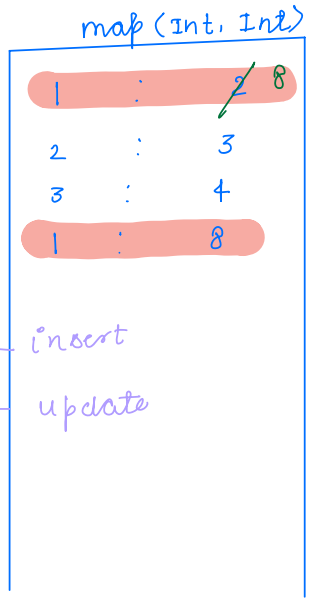
Value: population of each state

Map<String, Long>

Map<String, Map<String, Long>>

Hashset and hashset functionalities

Hashmap < Integer, Integer > map



1.) insert (key, value) O(1)

```
map.put(1, 2)
map.put(2, 3)
map.put(3, 4)
map.put(1, 8)
```

map.put(4, 8) — insert

map.put(2, 6) — update

2.) size() O(1) map.size() ⇒ 3

3.) delete (key) O(1) : map.remove(1)
map.size() = 2.

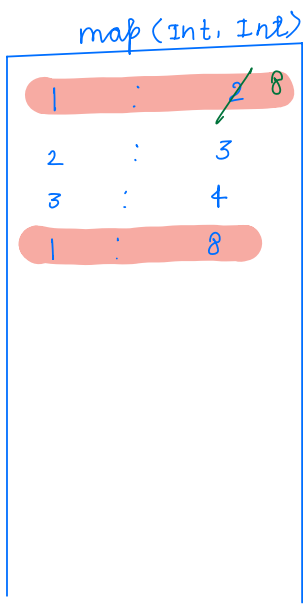
4.) update (key, value) O(1)

5.) search (key) O(1) :- Get value of a key.

map.get(2) = 3

h/w: map.get(8) = ?

- exception
- null



Hash set <Integer> [key ✓ , value X] — key is unique, key is unordered

1.) insert(key, value)

set.add(1)
set.add(2)
set.add(8)
set.add(2)

< 1 2 8 2 >

2.) size() set.size() = 3

3.) delete(key) set.remove(2) =
set.remove(10) = ?

4.) update(key, value) X

5.) search(key) : contains

< 1 2 8 2 >

set.contains(8) = true
set.contains(2) = false

Qu Given arr[n] and Q queries. find frequency of the elements provided in a query.

ip

A[] = | 2 | 6 | 3 | 8 | 2 | 8 | 2 | 3 | 8 | 10 | 6 |

Q = 4 | 2 | 8 | 3 | 5 |

Element	frequency
2	3
8	3
3	2
5	0

Brute force:

TC: $O(n^2)$

SC: $O(1)$

Idea 2: hashmap < Integer, Integer >

A[] = | 2 | 6 | 3 | 8 | 2 | 8 | 2 | 3 | 8 | 10 | 6 |

frequency of each element :-

Key : Array el - integer

value : freq - integer

Dry run for map creation

2	———	~~1~~ ~~2~~ 3
6	———	~~1~~ 2
3	———	~~1~~ 2
8	———	~~1~~ ~~2~~ 3
10	———	1

final map

2	———	3
6	———	2
3	———	2
8	———	3
10	———	1

array element

present in map

```
int freq = map.get(el);
freq = freq + 1;
map.put(el, freq);
```

not present in map

```
map.put(el, 1)
```

Algorithm

```
void printFreq(int[] A, int[] Q) {
    Map<Integer, Integer> map = new Hashmap<>();
    for(int el: A) {                                  ---- O(n)
        if(map.containsKey(el)) {
            int freq = map.get(el);                   -- O(1)
            freq+=1;
            map.put(el, freq);                        -- O(1)
        } else {
            map.put(el, 1);
        }
    }

    for(int el: Q) {
        print(map.get(el));
    }
}
```

TC: O(n)

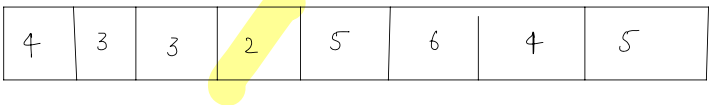
SC: O(n)

final map

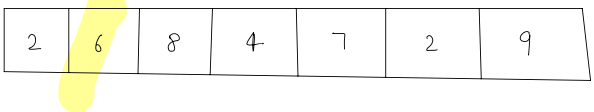
2	----	3
6	----	2
3	----	2
8	----	3
10	----	1

Qu2 Given arr[n], find first non-repeating element.

ans = 3

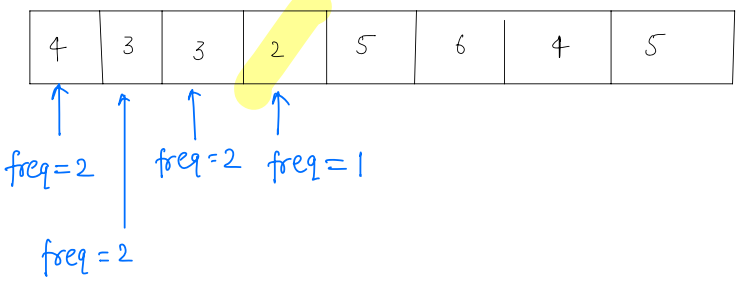


ans = 2



ans = 6

Brute force: $O(n^2)$, $O(1)$



map<El, freq> (Int, Int)

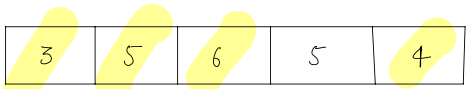
3 - 2
5 - 2
6 - 1
2 - 1
4 - 2

Code

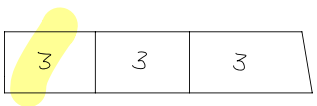
```
int firstNonRepeatingElement(int[] A) {
    Map<Integer, Integer> map = new Hashmap<>();
    for(int el: A) {                                  ——— O(n)
        if(map.containsKey(el)) {
            int freq = map.get(el);                   —— O(1)
            freq+=1;
            map.put(el, freq);                        —— O(1)
        } else {
            map.put(el, 1);
        }
    }
    for(int el: A){
        if(map.get(el) == 1) {
            return el;
        }
    }
    return -1;
}
```

TC: O(n)
SC: O(n)

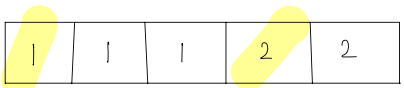
Qu3 Given arr[n], find count of distinct elements.



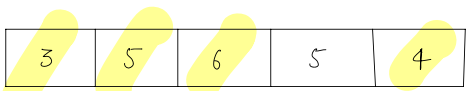
ans = 4



ans = 1



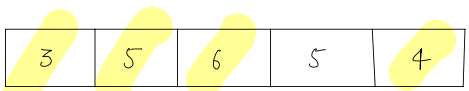
ans = 2



ans = 4

3 —— 1
5 —— 2
6 —— 1
4 —— 1

map.size() = Ans
↑
distinct elements.



set: < 3, 5, 6, 4 >

set.size() = 4.

Code

```
int countDistinctElements (int[] A) {
    Set<Integer> set = new HashSet<>();
    for(int el : A) {
        set.add(el);
    }
    return set.size();
}
```

TC: O(n)
SC: O(n)

Break: 8:32 - 8:42

Qu4 Given arr[n], check if there exists a subarray with a sum equal to zero.

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

Observations

	0	1	2	3	4	5
pf[] =	10	15	0	18	21	16
arr:	10	5	-15	18	3	-5

if(pf[i] == 0) — subarray (0-i) having sum = 0

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

if any el in pf[] is repeating, subarray having sum = 0.

pf[2] = 5

pf[8] = 5

~~arr[0] + arr[1] + arr[2]~~ = ~~5~~ — (I)

~~arr[0] + arr[1] + arr[2]~~ = ~~5~~ — (II)

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

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

(II) − (I)

arr[3] + arr[4] + arr[5] ..... arr[8] = 0

subarray (3, 8) = 0

Code:

```
boolean checkforsubarraysumAsZero(int[] arr){
    int[] ps = getPrefixsum(arr);
    set<Integer> set = new Hashset<>();
    for(int el: ps) {
        if(el==0) {
            return true;
        }
        set.add(el);
    }
    if(pf.length != set.size()){
        return true;
    }
    return false;
}
```

TC: O(n)
SC: O(n)

Thankyou ☺

Doubts