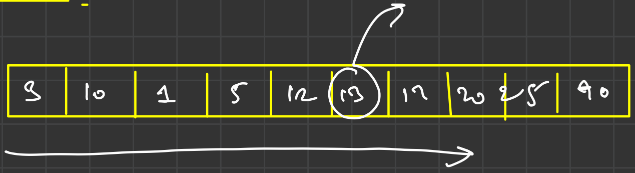
Hashing

# A technique useful for searching purpose.

1) Linear Search



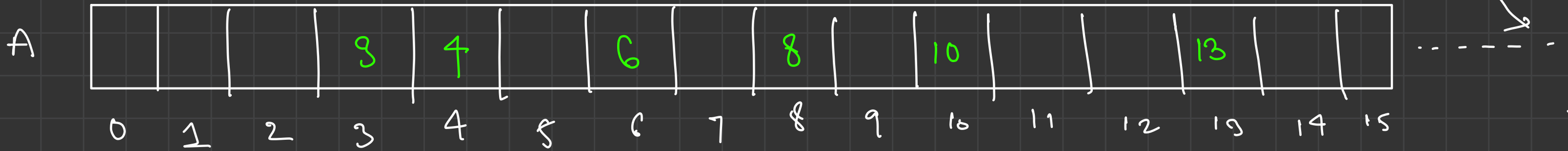
TC: O(N) Searching.

2) Binary Search $\longrightarrow$ sorted Array.

{TC: O(log N) Searching}

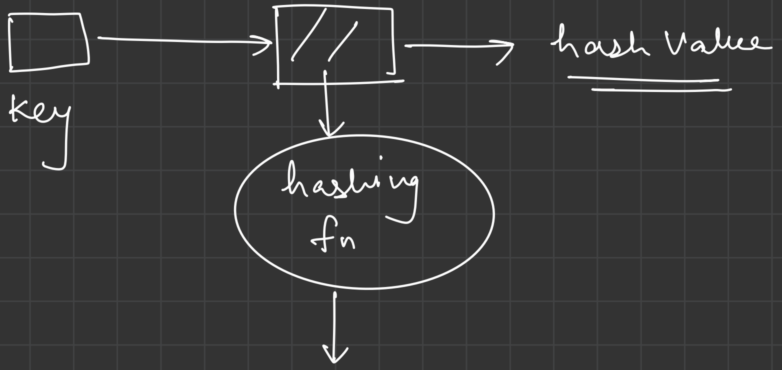
③ Hashing $\longrightarrow \overset{TC}{O(1)}$ Searching !

{ 8 , 3, 13 , 6, 4, 10 }, 50



key → ③ if (A[key] != NULL)
true; //present

TC: O(1) high Memory is required,
hence we introduced hashing.

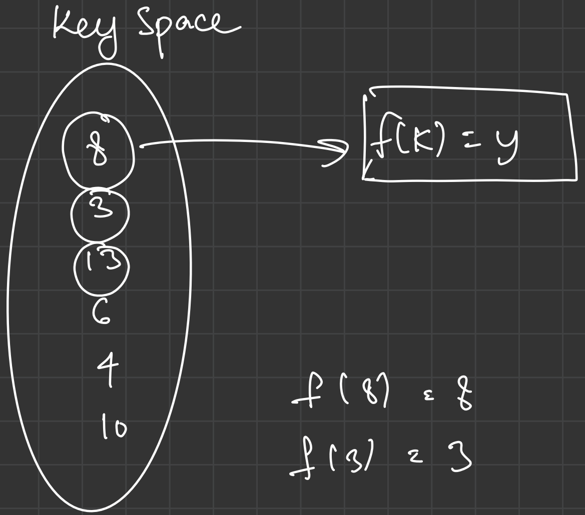


step1

$$\begin{tikzcd} g(x) \arrow[d] & = & K \arrow[d] \\ \boxed{key} \arrow[rr] & & \text{integer value.} \end{tikzcd}$$

step2

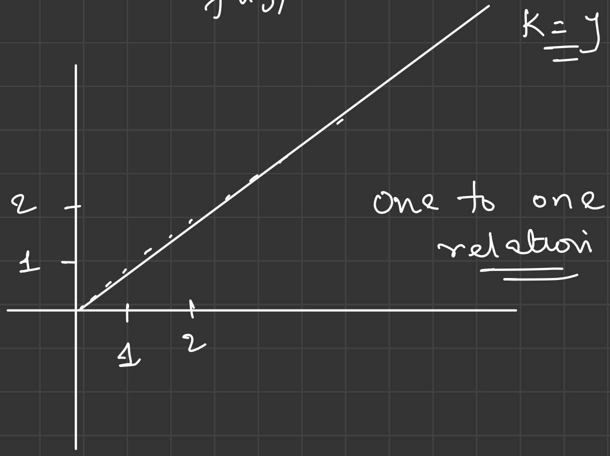
$$\begin{tikzcd} f(K) \arrow[d] \arrow[rr, bend right] & = & y \\ \underline{\text{hashing fn}} & & \end{tikzcd}$$



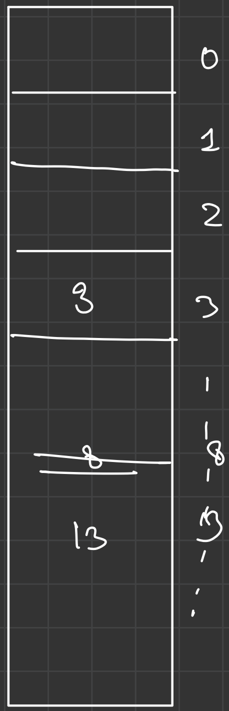
$f(8) = 8$

$f(3) = 3$

$f(13) = 13$



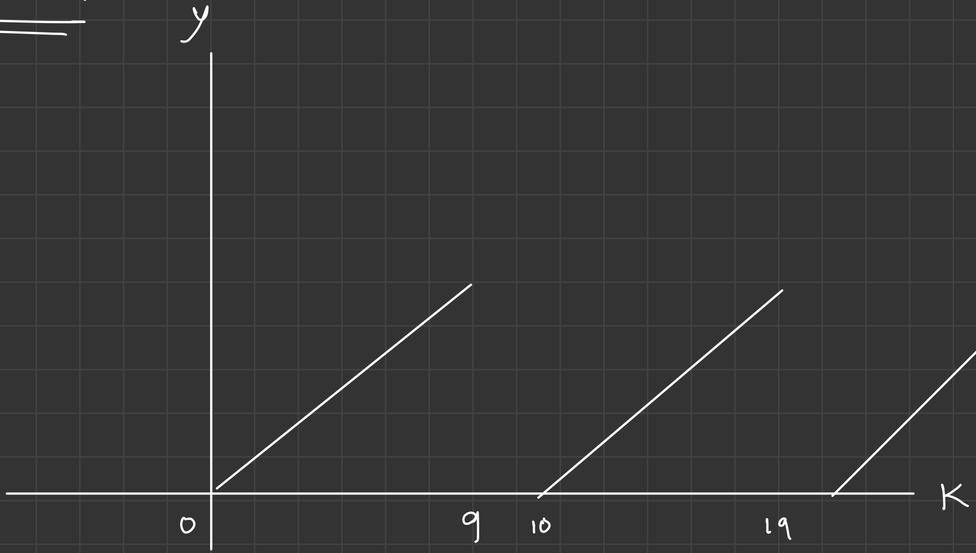
One to one relation

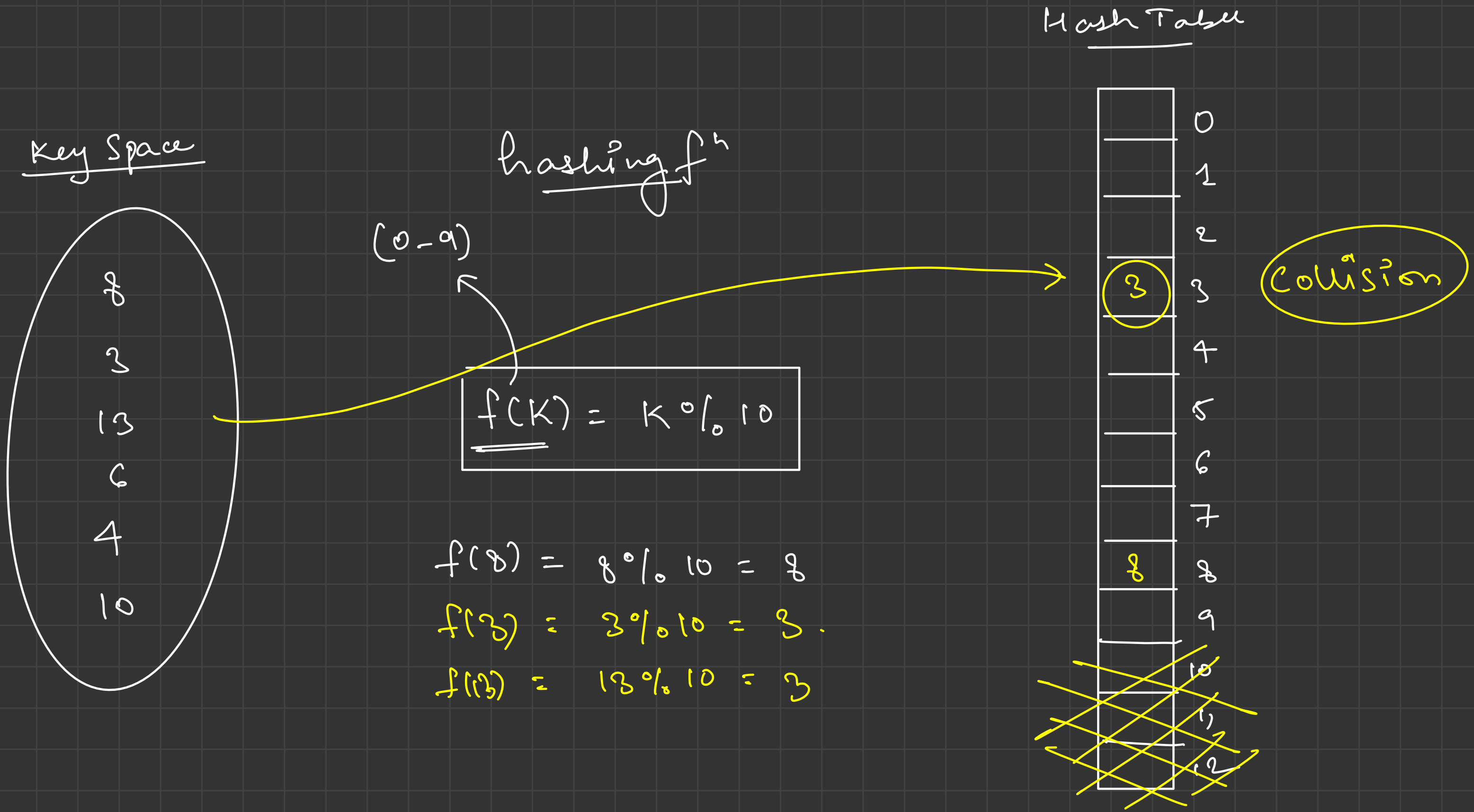


HashTable

Many to One Relation

$y = K \% 10$





$f(8) = 8 \% 10 = 8$

$f(3) = 3 \% 10 = 3$.

$f(13) = 13 \% 10 = 3$

$$f(K) = \underline{\underline{K \% \boxed{20}}}$$

$\downarrow$

size of $\underline{\underline{\text{hashtable}}}$

if hashtable size $= \underline{\underline{1000}}$

✓

$$\boxed{f(K) = K \% \,1000}$$

Methods to Remove Collision.

$$\begin{tikzcd}
 & \text{Methods to Remove Collision} \arrow[dl] \arrow[dr] & \\
\text{Open Hashing} \arrow[d] & & \text{Closed Hashing} \\
\text{chaining} & &
\end{tikzcd}$$

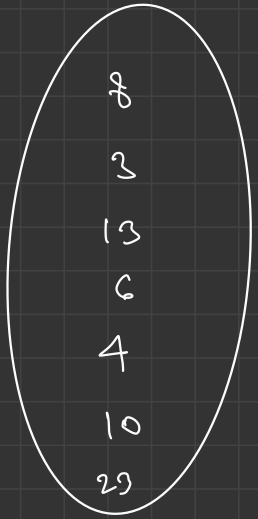
Closed Hashing

→ Linear Probing

→ quadratic Probing.

Chaining

Key Space



hashing f^n

(0-9)

$$\boxed{f(K) = K \% 10}$$

$f(8) = 8$

$f(3) = 3$

$f(13) = 3$

$f(6) = 6$

$f(4) = 4$

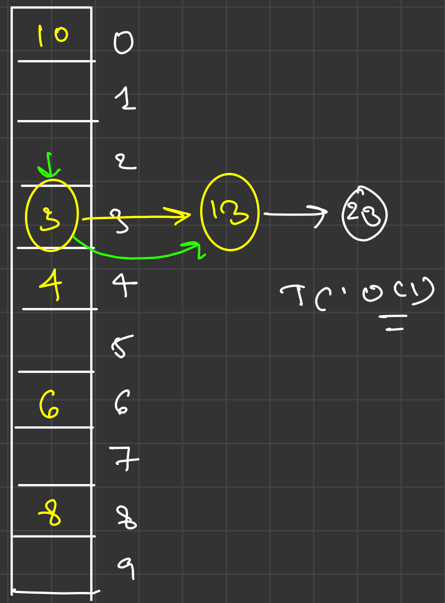
$f(10) = 0$

$f(23) = 3$

find (13)

3

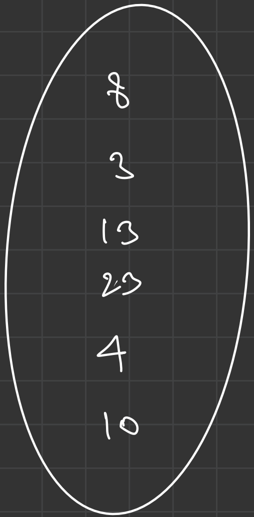
Hash Table



T.C: O(1)

Linear Probing

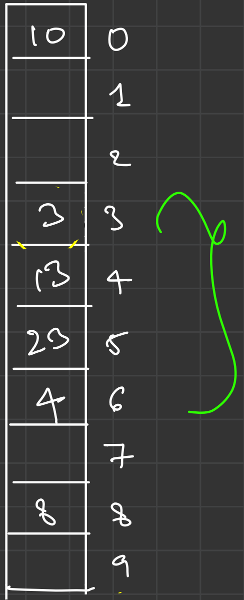
Key Space



hashing f^n

$$\boxed{f(K) = K \% 10}$$

Hash Table



$$\boxed{f'(K) = [f(K) + h(i)] \% 10}$$

$h(i) = i$, $i = 0, 1, 2 \ldots$

find(4)

$f'(8) = [f(8) + h(0)] \% 10 = (8 + 0) \% 10 = 8$

$f'(3) = [f(3) + h(0)] \% 10 = (3 + 0) \% 10 = 3$

$f'(13) = [f(13) + h(0)] \% 10 = (3 + 0) \% 10 = 3$

$f'(13) = [f(13) + h(1)] \% 10$

$= (3 + 1) \% 10 = 4$

Quadratic Probing

$$f(K) = K \% \text{size}$$

$$f'(K) = [f(K) + h(i)] \% \text{size}$$

$$h(i) = i^2 \quad , \quad i = 0, 1, 2, 3 \ldots$$

8
3
13
23
4
10

size = 10

$f'(8) = [8 + 0] \% 10 = 8$ ✓

$f'(3) = [3 + 0] \% 10 = 3$ ✓

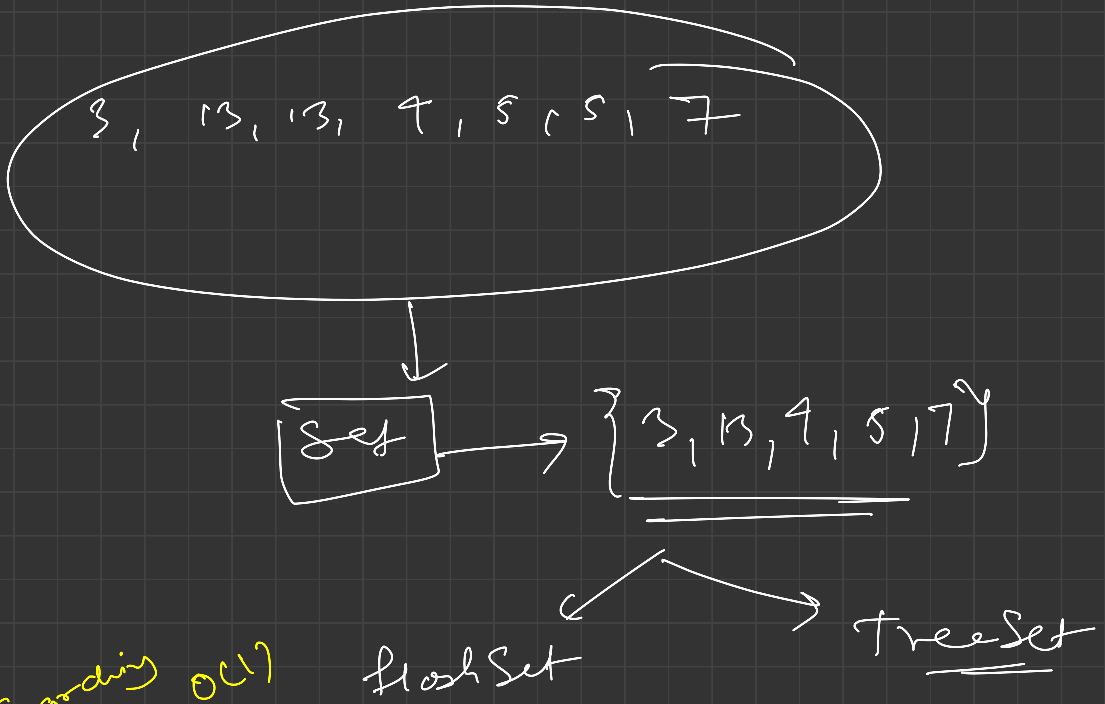
$f'(13) = [3 + 1] \% 10 = 4$ ✓

$f'(23) = [3 + 4] \% 10$

$= 7 \% 10 = \boxed{7}$ ✓

$$\begin{tikzcd}
\text{HashMap, HashSet} \arrow[d] & \boxed{\text{TreeMap, TreeSet}} \arrow[d] \\
\{\text{Hashing Algo}\} & \{\text{Red - Black trees}\}
\end{tikzcd}$$

Sets Collection of unique entities



Searching
insertion O(1)

Values will be in
Random order

Searching
insertion O(logN)

Values are sorted
in asc. order

HashMap

→ Key - value pairs

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

Key	Val
1	Ritik
2	Aca'o

→ Keys are in Random order

→ offers O(1) searching, O(1) insertion.

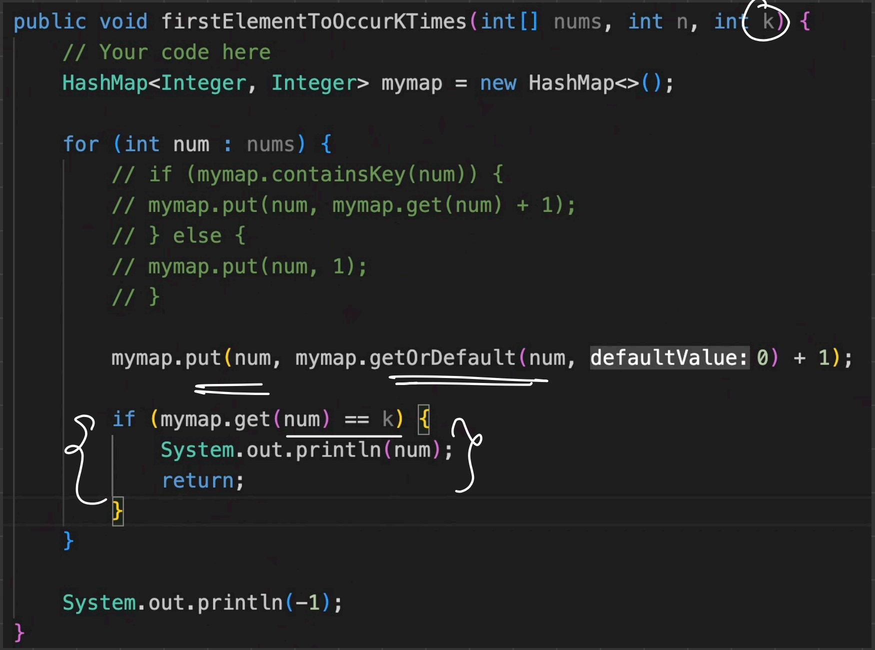
TREEMAP

→ stores keys in asc. order.

→ offers $O(\log N)$ search, $O(\log N)$ insertion.

[1, 7, 4, 3, 4, 8, 7]

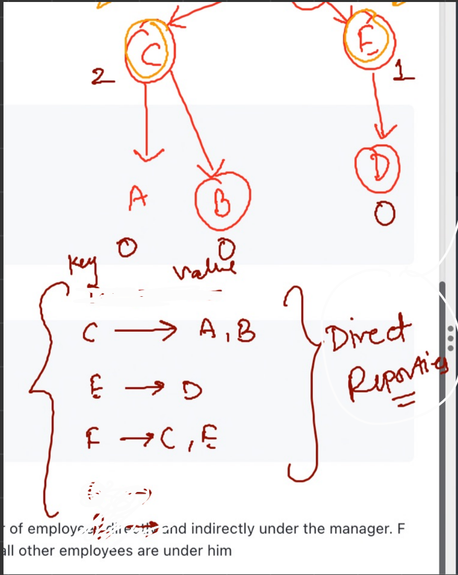
K = 2



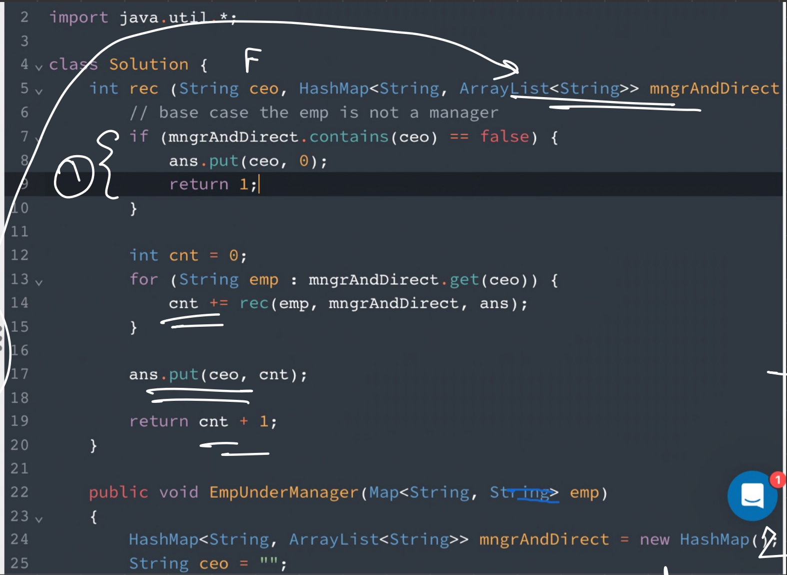
```java
public void firstElementToOccurKTimes(int[] nums, int n, int k) {
    // Your code here
    HashMap<Integer, Integer> mymap = new HashMap<>();

    for (int num : nums) {
        // if (mymap.containsKey(num)) {
        // mymap.put(num, mymap.get(num) + 1);
        // } else {
        // mymap.put(num, 1);
        // }

        mymap.put(num, mymap.getOrDefault(num, defaultValue: 0) + 1);

        if (mymap.get(num) == k) {
            System.out.println(num);
            return;
        }
    }

    System.out.println(-1);
}
```

Key	Value
1	1
7	1
4	~~1~~ 2
3	1

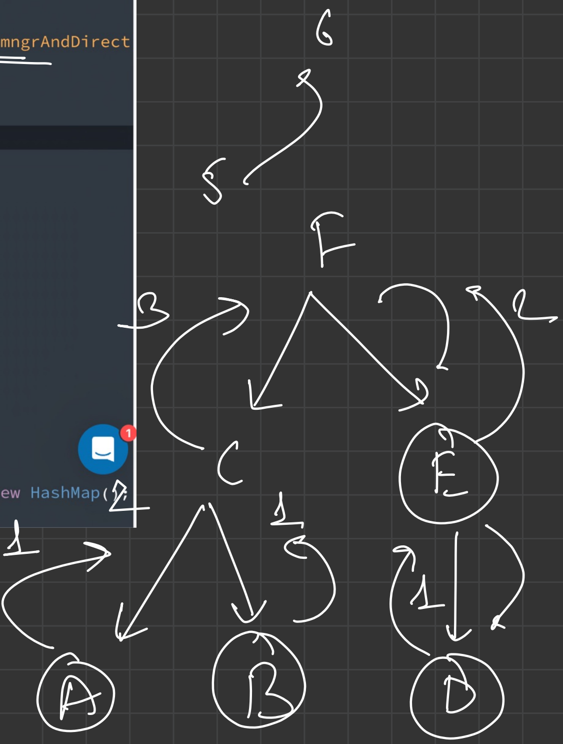
TC : O(N)



of employees directly and indirectly under the manager. F
all other employees are under him



```
import java.util.*;

class Solution {
    int rec (String ceo, HashMap<String, ArrayList<String>> mngrAndDirect
        // base case the emp is not a manager
        if (mngrAndDirect.contains(ceo) == false) {
            ans.put(ceo, 0);
            return 1;
        }

        int cnt = 0;
        for (String emp : mngrAndDirect.get(ceo)) {
            cnt += rec(emp, mngrAndDirect, ans);
        }

        ans.put(ceo, cnt);

        return cnt + 1;
    }

    public void EmpUnderManager(Map<String, String> emp)
    {
        HashMap<String, ArrayList<String>> mngrAndDirect = new HashMap(
        String ceo = "";
```



$A \longrightarrow 0$

$B \longrightarrow 0$

$C \longrightarrow 2$

$D \longrightarrow 0$

$E \longrightarrow 1$

$F \longrightarrow 5$