longest subarray with equal freq. of 0's, 1's & 2's

	0	1	2	3	4	5	6	7	8	9	10	11	12
	1	1	2	0	1	0	1	2	1	2	2	0	1
0	0	0	0	0	1	1	2	2	2	2	2	2	3	3
1	0	1	2	2	2	3	3	4	4	5	5	5	5	6
2	0	0	0	1	1	1	1	1	2	2	3	4	4	4
Δ_{10}	0	1	2	2	1	2	1	2	2	3	3	3	2	3
Δ_{21}	0	-1	-2	-1	-1	-2	-2	-3	-2	-3	-2	-1	-1	-2

TC : O(N)
SC : O(N)

$\Delta_{10} \# \Delta_{21}$

0#0

1#-1

1#-1

2#-2

String	Integer
3#-2	9
0#0	-1
1#-1	0
3#-1	10
2#-2	1
2#-1	2
1#-2	5
2#-3	6
3#-3	8

1#-2

maxlen = ~~0~~ ~~2~~ ~~6~~ 9

$\Delta_{10} = f_1 - f_0$

$\Delta_{21} = f_2 - f_1$

11 - 2 = 9

12 - 9 = 3

7 - 1 = 6

4 - 1 = 3

ind - 0

3 - 0 = 3

$$\begin{cases} \Delta_{10} = x_1 - x_0 \\ \Delta_{21} = x_2 - x_1 \end{cases}$$

$$\begin{cases} \Delta'_{10} = x'_1 - x'_0 \\ \Delta'_{21} = x'_2 - x'_1 \end{cases}$$

Suppose $\underline{\underline{x_1 + m = x_1'}}$

$$\Delta_{10} = \Delta'_{10}$$

$$x_1 - x_0 = x_1' - x_0'$$

$$\cancel{x_1} - x_0 = (\cancel{x_1} + m) - x_0'$$

$$\boxed{x_0' = m + x_0}$$

$$\Delta_{21} = \Delta'_{21}$$

$$x_2 - x_1 = x'_2 - x'_1$$

$$x_2 - x_1 = x'_2 - (x_1 + m)$$

$$x_2 - \cancel{x_1} = x_2' - \cancel{x_1} - m$$

$$\boxed{x_2 + m = x_2'}$$

	0	1	2	3	4	5	6
arr	0	1	0	2	0	1	0
0 →	1	1	2	2	3	3	4
1 →	0	1	1	1	1	2	2
2 →	0	0	0	1	1	1	1
Δ_{10} →	-1	0	-1	-1	-2	-1	-2
Δ_{21} →	0	-1	-1	0	0	-1	-1

	Code	index
-1#-1	0#0	-1
-1#0	-1#0	0
0#-1	0#-1	1
-1#-1	-1#-1	2
-1#0	-2#0	4
-2#0	-2#-1	6
-2#-1		

$3 - 0 = 3$

$5 - 2 = 3$

```java
public static int longestSubarray(int[] arr) {
    // write code here
    HashMap<String, Integer> map = new HashMap<>();
    map.put(key: "0#0", -1);

    int count0 = 0;
    int count1 = 0;
    int count2 = 0;

    int maxLen = 0;  3

    for (int i = 0; i < arr.length; i++) {
        int num = arr[i];
        if (num == 0) {
            count0++;
        } else if (num == 1) {
            count1++;
        } else if (num == 2) {
            count2++;
        }

        int d10 = count1 - count0;
        int d21 = count2 - count1;

        String code = d10 + "#" + d21;
        if (map.containsKey(code)) {
            int len = i - map.get(code);
            maxLen = Math.max(maxLen, len);
        } else {
            map.put(code, i);
        }
    }

    return maxLen;
}
```

Rabbit in forest

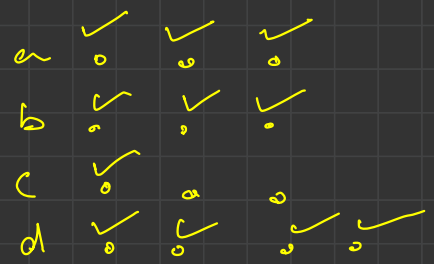
	0	1	2	3	4	5	6	7	8
arr[] =	2	2	3	1	0	2	2	3	1

a	0	1	5
b	2	7	
c	3	8	
d	4		
e	6		

7 rabbit say

14	2
7	3
2	4
3	0
	1

2, 2, 2, 2, 2, 2, 2, 3, 3, 3, 3, 3
3, 3, 3, 3, 3, 3, 3, 3, 3, 4, 4, 4, 4
4, 4, 4, 0, 0, 1, 1, 1



```java
class Solution{
    public int numRabbits(int[] arr) {
        // write code here
        HashMap<Integer, Integer> fmap = new HashMap<>();
        for (int a : arr) {
            fmap.put(a, fmap.getOrDefault(a, defaultValue: 0) + 1);
        }

        int ans = 0;
        for (int key : fmap.keySet()) {
            int gs = key + 1;
            int reportees = fmap.get(key);

            int ng = reportees / gs;
            if (reportees % gs > 0) {
                ng++;
            }
            ans += ng * gs;
        }

        return ans;
    }
}
```

= 2/4 = 0

2

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

a	freq
2	4
3	2
1	2
0	1

gs = 1

reportee = 1

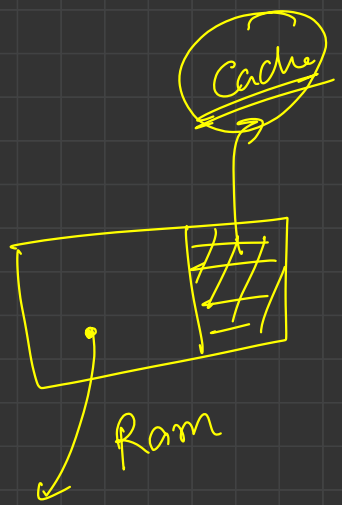
ng = 1/1 = 1

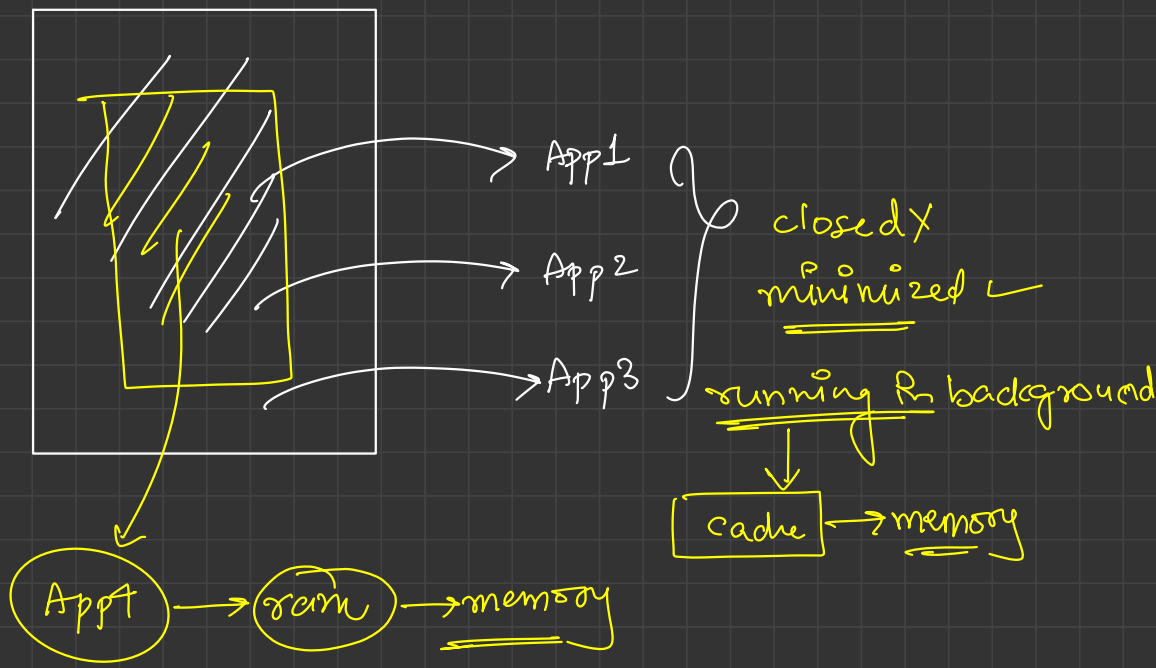
1%1 = 0

ans = 6 + 4 + 2 + 1

= 13

LRU Cache


Cache
Ram


App1
App2
App3
closed X
minimized
running in background
cache
memory
App1
ram
memory

cache memory → fixed size

↓

hence a limit of App running in background.

manages cache memory ← OS

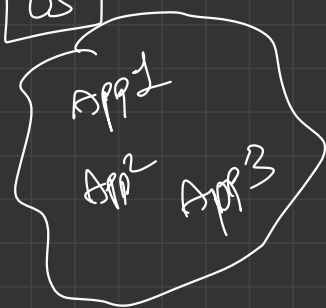
limit

↓

3 Applications

↓

max^m in cache memory!



cache memory

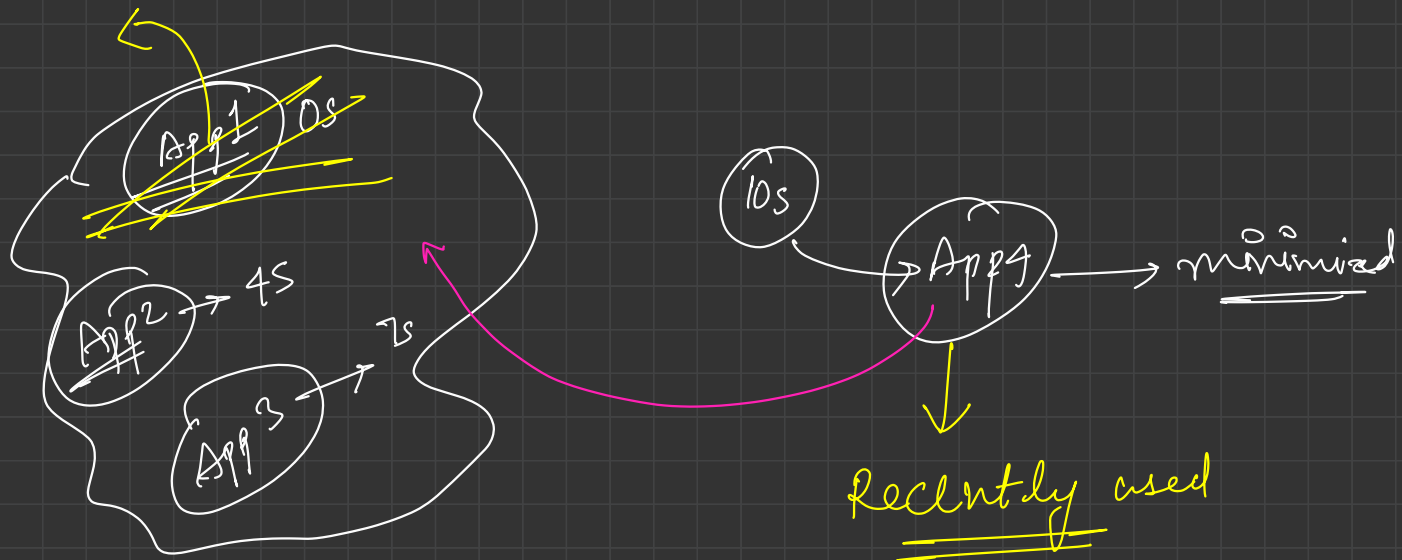
App1 → minimized

App2 → minimized

App3 → minimized

App4 → minimized

to manage Cache Memory → LRU {least recently used}
→ LFU {least freq. used}



Cache Memory

Steps

1. App1 → opened 0s
2. App2 → opened 4s
3. App3 → opend 7s
4. App1 → Read 10s
5. App4 → opened 13s
6. App5 → opened 14s
7. App4 → Read 15s

opened → cache memory entry

capacity = 3

App1 ~~0s~~ 10s

~~App2 4s~~ → removed

~~App3 7s~~ → removed

App4 15s

App5 14s

App1 0s

App2 4s

App3 7s

Cache Memory

```
class LRUCache {
    // your code here
    public LRUCache(int capacity) {
        // your code here
    }

    public int get(int key) {
        // your code here
    }

    public void set(int key, int value) {
        // your code here
    }

}
```

max application in cache memory at a time.

move this app to most recently used.

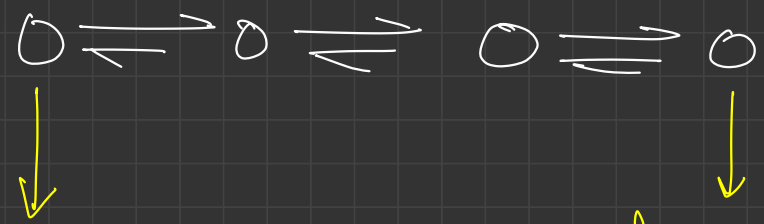
application id

value

Cache Memory = 3

4 → 50 12s
2 → 30 6s
3 → 70 10s

1. set(1, 10)
2. set(2, 30)
3. set(3, 40)
4. get(2) → 30
5. set(4, 50)
6. set(3, 70)
7. get(4) → 50



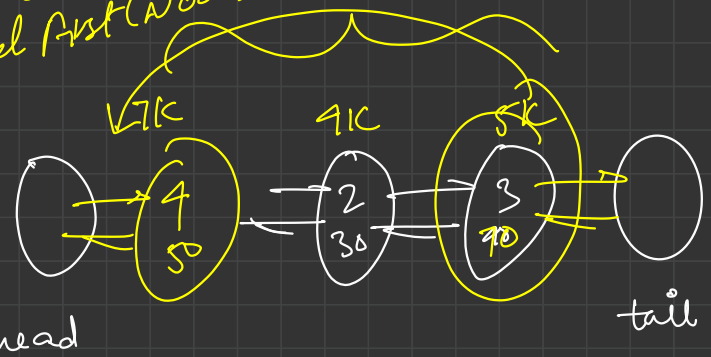
Most Recently used person

least Recently used person

① set(1, 10)
② set(2, 30)
③ set(3, 40)
④ get(2) → 30
⑤ set(4, 50)
⑥ set(3, 70)
⑦ get(4) → 50

addFirst(Node)

addFirst(Node)



☆ Always add in front

Because it is most recently used

☆ LRU APP

tail.prev

id	Address
2	4K
3	5K
4	7K

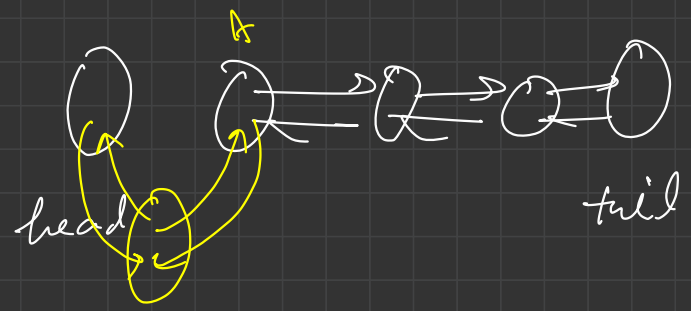
size of map == capacity (=3)

hence cache memory is full

Remove LRU

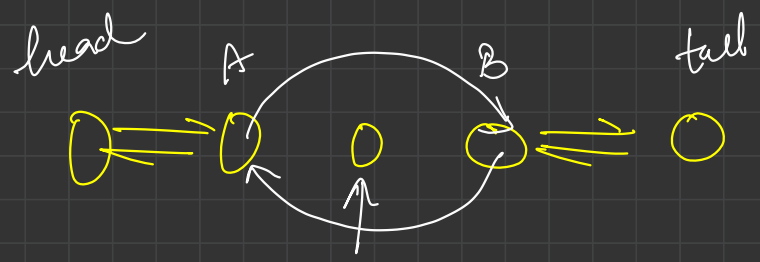
① Remove(Node)
② addFirst(Node)

moveTofront()




```
Node A = head.next
node.next = A
node.prev = head
A.prev = node
head.next = node;
```

Remove Node



Node A = node.prev

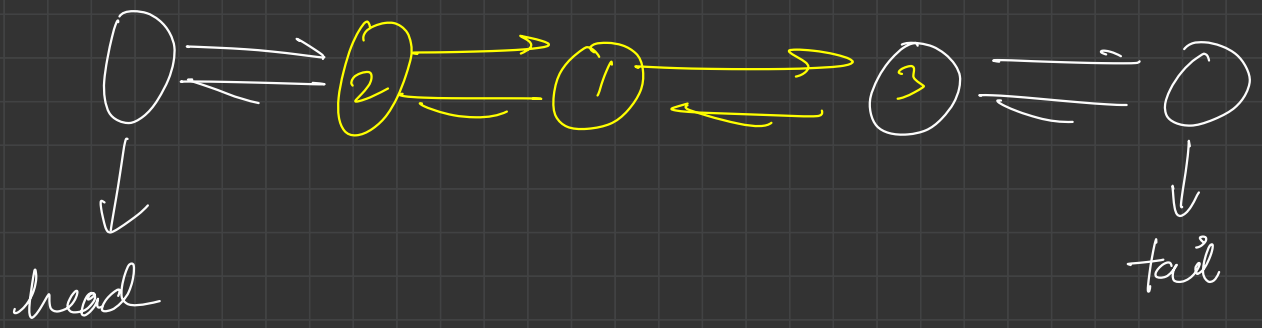
Node B = node.next

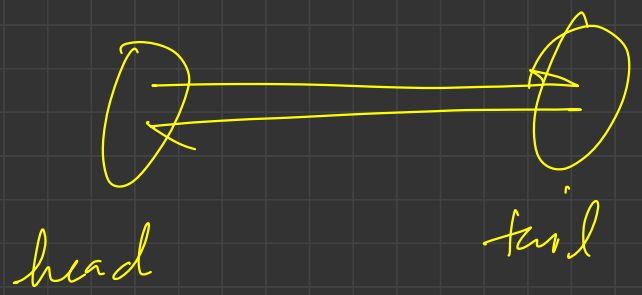
A.next = B

B.prev = A

node.next = null

node.prev = null


2
1
3
head
tail


head
tail

Snapshot Array

0	1	2	3	4	5	6	7
20	0	4	5	0	10	0	2

snap_id = ~~0~~ ~~1~~ 2

```
public static void SnapshotArray(int length) {
}

public static void set(int index, int val) {

}

public static int snap() {

}

public static int get(int index, int snap_id) {

}
```

get(3,1) → 10

get(3,0) → 5

set(3, 10)

set(5, 10) set(7, 2) set(0, 20)

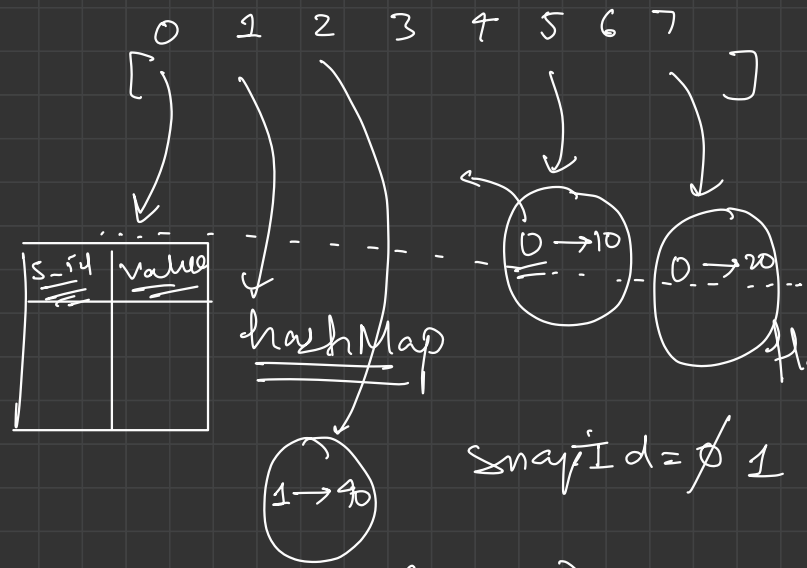
Snapshots

0

0	1	2	3	4	5	6	7
0	0	4	5	0	0	0	0

1

0	1	2	3	4	5	6	7
20	0	4	10	0	10	0	2



[0, ∞]

int[]

HashMap<Integer, Integer>[ ] ds

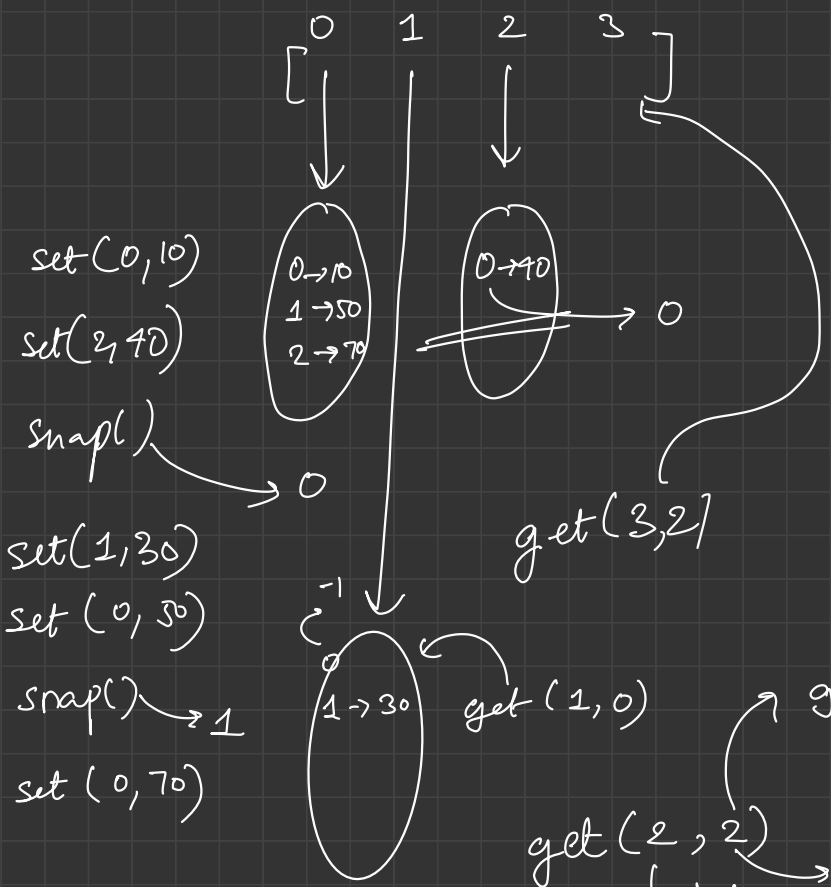
snapId = ~~0~~ 1

Set (5, 10)

Set (7, 20)

mapArray[idx].put(snap_id, value)

set (2, 40



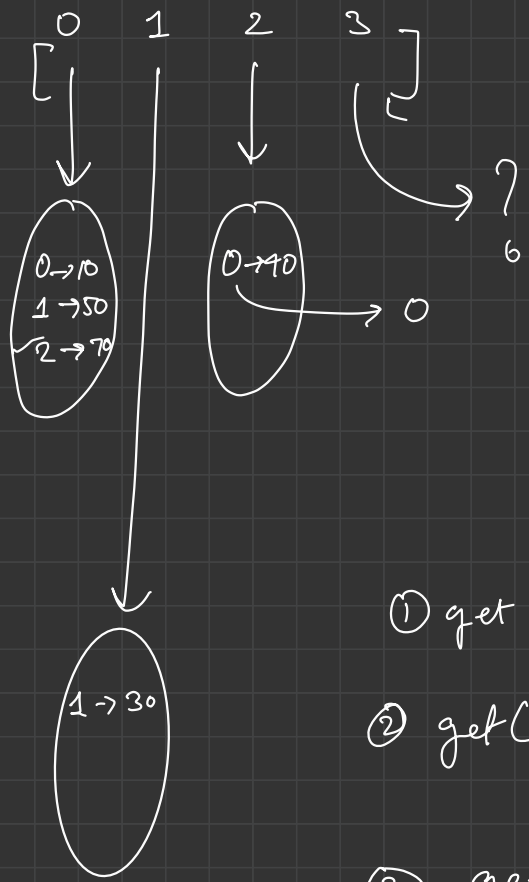
snapId = ~~0~~ ~~1~~ 2

snap 0	[10, 0, 40, 0]
snap 1	[50, 30, 40, 0]
snap 2	[70, 30, 40, 0]

goto last snapId where it was updated.

get(2, 2)
↳ index
↳ snap_id

```
while(snap_id >=0 && mapArray[index].containsKey(snap_id) == false
{ snapId--; }
```




```java
public static int get(int index, int snap_id) {
    if (mapArray[index] == null) {
        return 0;
    }

    // get the last snap_id where this mapArray[index] got updated
    while (snap_id >= 0 && mapArray[index].containsKey(snap_id) == false) {
        snap_id--;
    }

    if (snap_id == -1) {
        return 0;
    }

    return mapArray[index].get(snap_id);
}
```

① get(0, 2) → 70

② get(1, 0) → 0

-1

③ get(2, 4) → 40

~~3~~ ~~2~~ ~~1~~ 0

④ get(3, 2) → 0