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

arr[] = { 1, 1, 2, 0, 1, 0, 1, 2, 1, 2, 2, 0, 1 }

$1 \rightarrow x$
$2 \rightarrow y$
$0 \rightarrow z$

$(freq_1 - freq_2) \# (freq_2 - freq_0)$

$(x - y \# y - z)$

$1 \rightarrow x + a$
$2 \rightarrow y + a$
$0 \rightarrow z + a$

$(x+m) - (y+m) \# (y+m) - (z+m)$

$(x - y \# y - z)$

```java
static int countEqualSubarray01(int arr[], int n) {
    // Write your code here
    // Map -> code, index
    // code -> difference of freq1 - freq2 # freq2 - freq0
    HashMap<String, Integer> mymap = new HashMap<>();
    mymap.put(key: "0#0", -1);

    int maxLen = 0;

    int cnt0 = 0;
    int cnt1 = 0;
    int cnt2 = 0;

    for (int i = 0; i < n; i++) {
        if (arr[i] == 0) {
            cnt0++;
        } else if (arr[i] == 1) {
            cnt1++;
        } else if (arr[i] == 2) {
            cnt2++;
        }

        String code = (cnt1 - cnt2) + "#" + (cnt2 - cnt0);
        if (mymap.containsKey(code) == true) {
            int len = i - mymap.get(code);
            maxLen = Math.max(len, maxLen);
        } else {
            mymap.put(code, i);
        }
    }

    return maxLen;
}
```

arr[] = { 1, 1, 2, 0, 1, 0, 1, 2, 1, 2, 2, 0, 1 }

(x, y, z) (x+m, y+m, z+m)

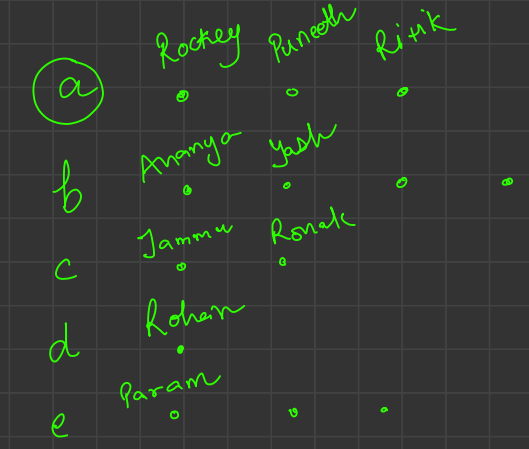
TC: O(N)

SC: O(N)

Rabbits in forest

rabbits with different colours

arr[] = { 2, 2, 3, 1, 0, 2, 2, 3, 1 }



freq

same colour → repeatees

2 (3) → 4

3 (4) → 2

1 (2) → 2

0 (1) → 1

no. of grps

2 + gs

1

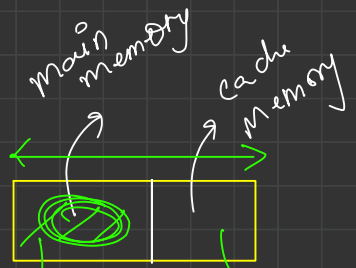
1

1

total

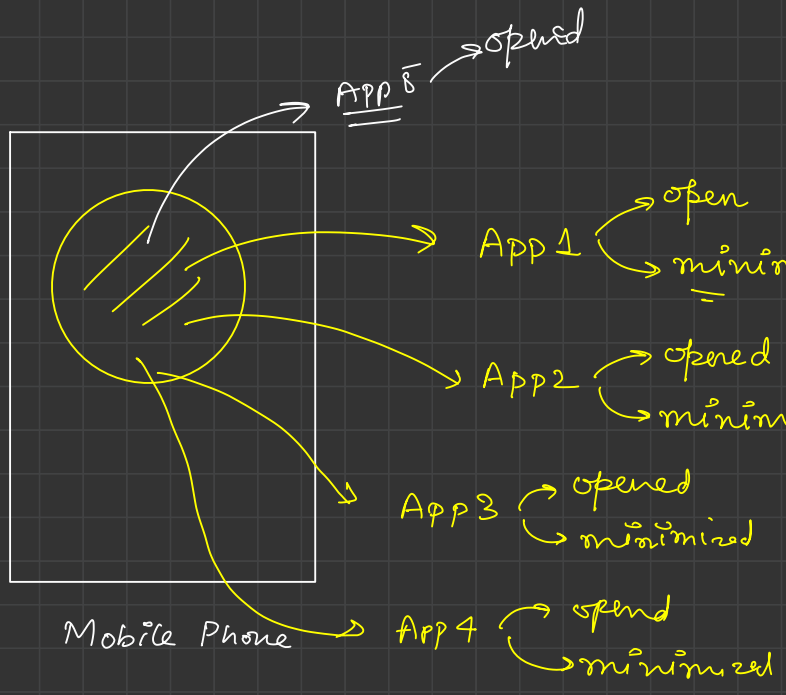
{ min rabbits } continued.

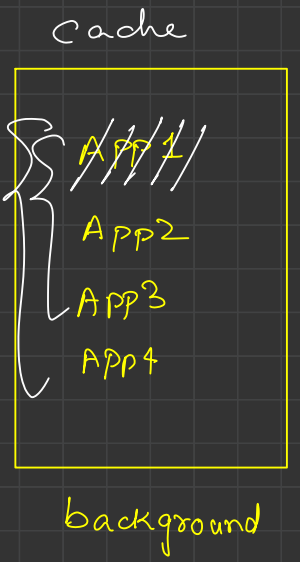
LRU Cache



Opened Application runs in the main memory

background work is done cache memory

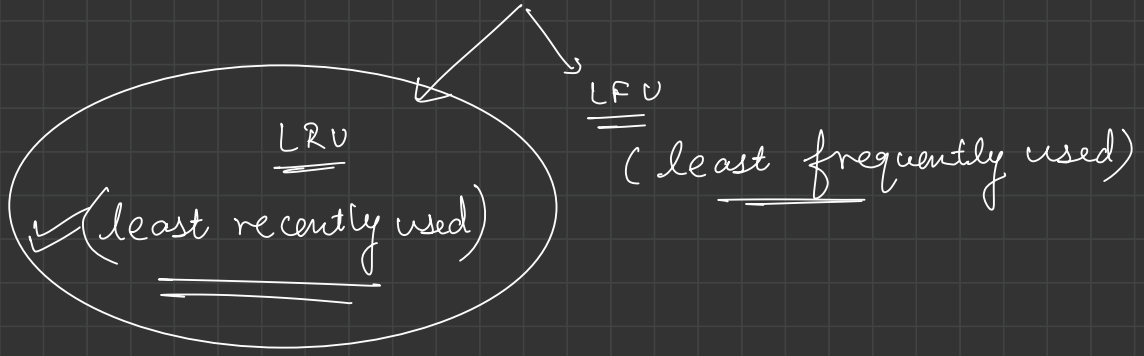




- ✱ limited Cache Memory
- ✱ Can't have Infinite Applications in background
- ✱ there is some limit of Number of application to running in background.

lets limit = 3 application

Removal of application when cache memory reaches its limits



depending upon OS you are using cache memory uses one of them.

Opened → Cache Memory

Step

0s App1 → opened

2s App2 → opened

5s App3 → opened

9s App4 → opened

10s App2 → Read

12s App5 → opened

Mobile phone

Capacity = 3

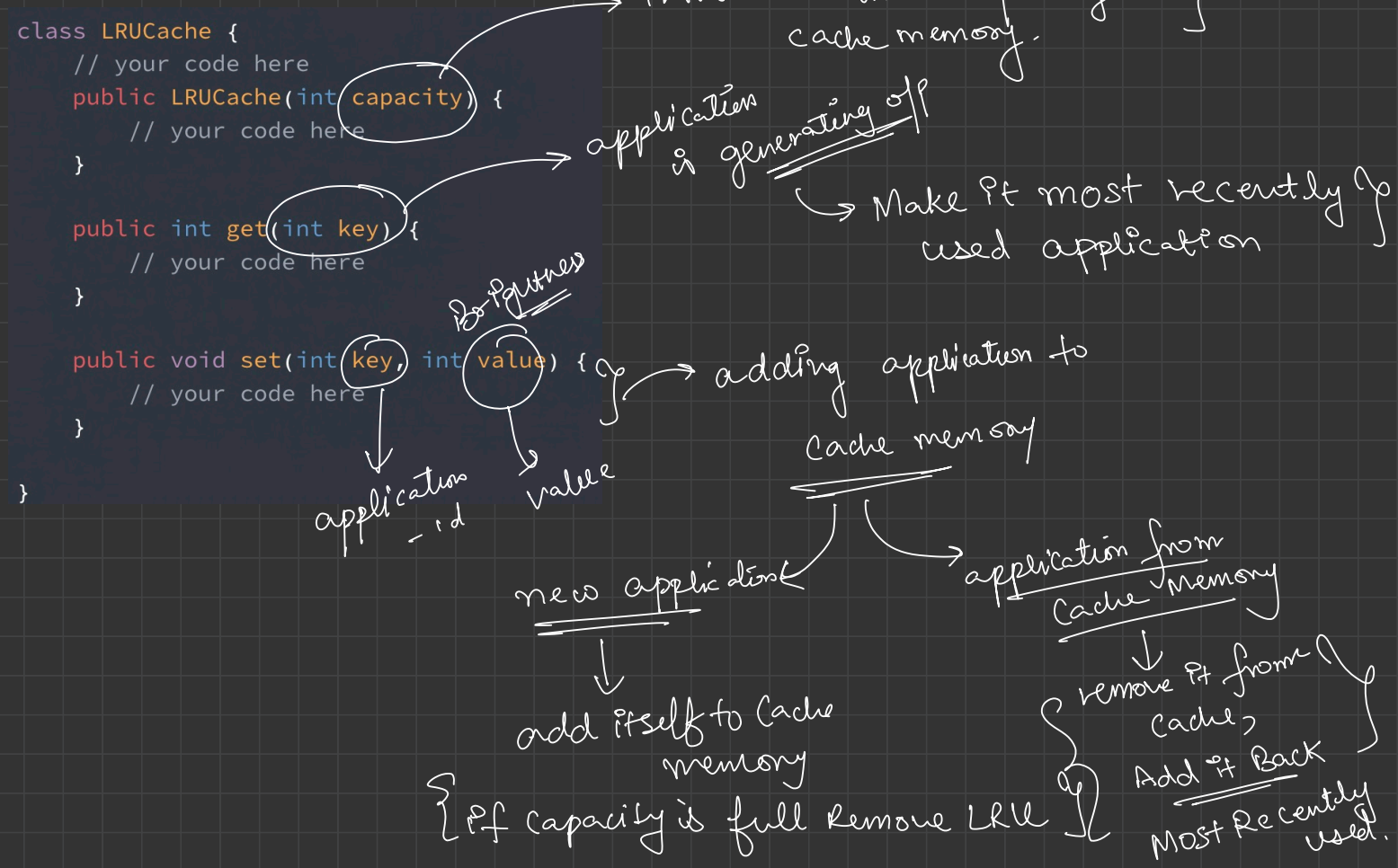
App2 → 10s

App5 → 12s

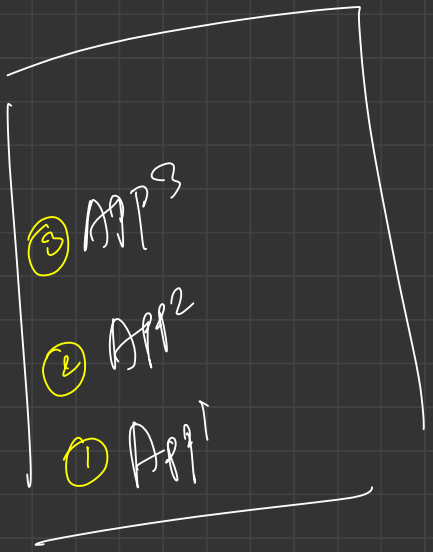
App4 → 9s

Cache Memory

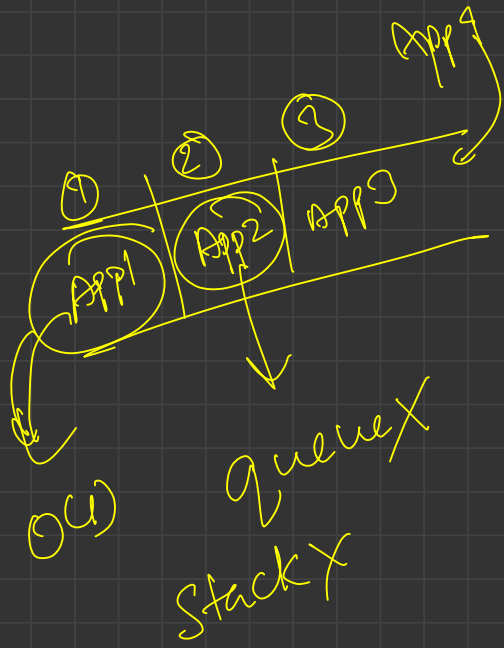
(LRU)


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
}
}
Initialize max^m capacity to cache memory.
application is generating o/p
Make it most recently used application
Do figures
adding application to Cache memory
application -id
value
new application
application from Cache memory
add itself to Cache memory
remove it from cache
Add it Back
Most Recently used.
{ If capacity is full Remove LRU

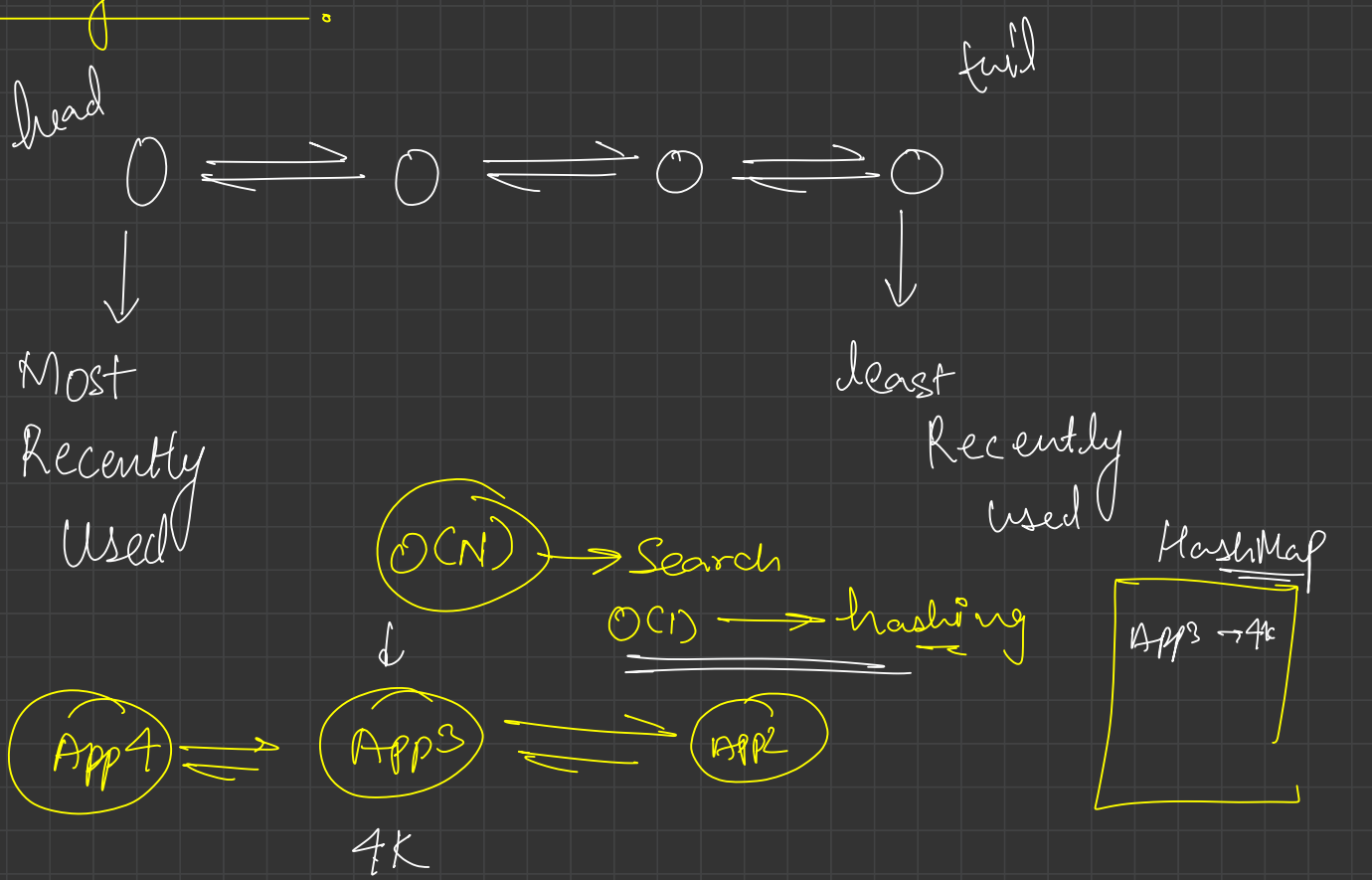
data structure suitable for this?



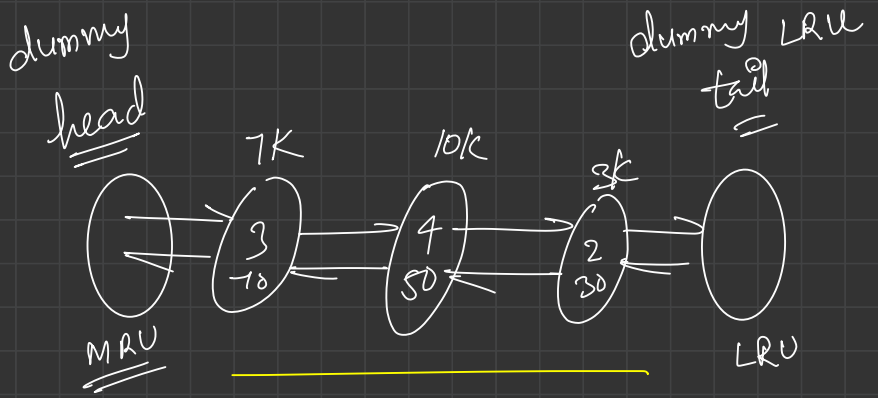
Cache Memory



Doubly Linked List



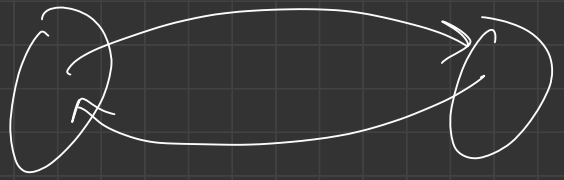
① set (1, 10)
② set (2, 30)
③ set (3, 40)
④ get (2)
⑤ set (4, 50)
⑥ set (3, 70)
⑦ get (4)

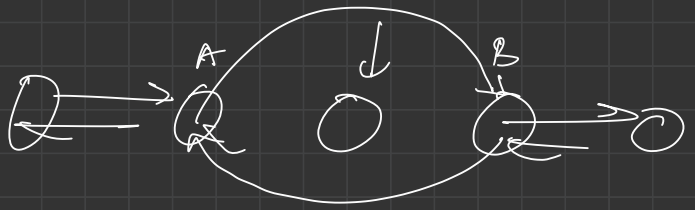


Id	Addl
2	3K
4	10K
3	7K

- Remove Node( )
- Add front ( )
- move Node front ( )

* LRU app
↳ tail. prev



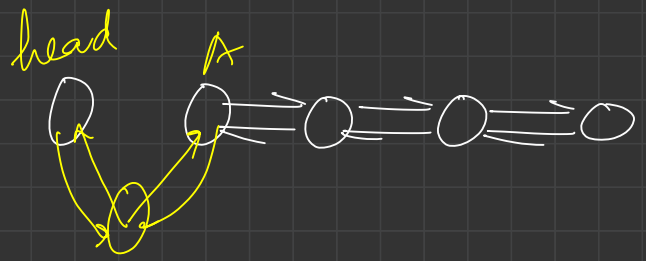


Node A = node.prev
Node b = node.next

A.next = B

B.prev = A

Node.next = null
Node.prev = null



Node A = head.next;

node.next = A

node.prev = head

head.next = node

A.prev = node

Snapshot Array

```
   0  1  2  3  4  5  6  7
[  0, 5, 4, 0, 5, 0, 0, 0 ]
```

snapid = ~~0~~ ~~1~~ 2

set (1, 3) } Snapshot 0
set (4, 5) }

snap()

set (1, 5) } Snapshot 1
set (2, 4) }

snap()

get (1, 0) → 3 }
get (1, 1) → 5 }

snapid = 0

```
   0  1  2  3  4  5  6  7
[  0, 3, 0, 0, 5, 0, 0, 0 ]
```

snapid = 1

```
   0  1  2  3  4  5  6  7
[  0, 5, 4, 0, 5, 0, 0, 0 ]
```

snapId = 0

```
static HashMap<Integer, Integer>[] mapArray;
static int snapId;

public static void SnapshotArray(int length) {
    mapArray = new HashMap[length];
    snapId = 0;
}

public static void set(int index, int val) {
    if (mapArray[index] == null) {
        mapArray[index] = new HashMap<>();
    }

    mapArray[index].put(snapId, val);
}

public static int snapId
    int val = snapId;
    snapId += 1;
    return val;
}

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

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

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