- Minimum window substring
- Longest subarray with equal freq of 0's, 1's, 2's.
- LRU Cache

Minimum window Substring.

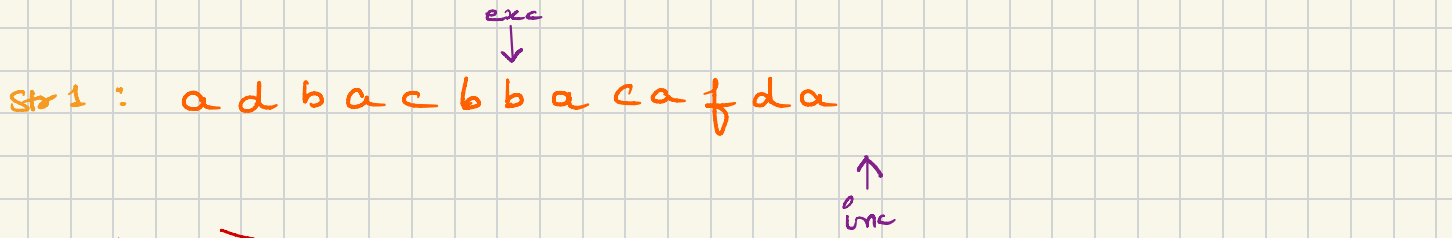
str1 : a d b a c b b a c a f d a

str2 : c b a b d

Brute Force

- generate all the substring, try to match min freq of each char in substring from str2.

TC : $O(N^2 \times M)$	SC : $O(N)$

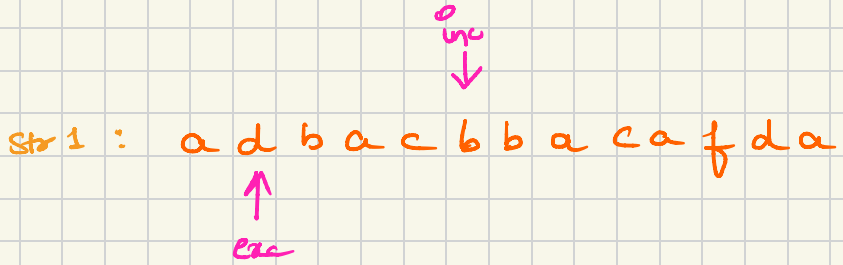


TC : O(M + N*26)

≈ O(M+N)

SC : O(1)

Len = ~~6~~ 5



str2 : c b a b d

freqMap2

c ⟶ 1
b ⟶ 2
a ⟶ 1
d ⟶ 1

mcnt = ~~1 2 3~~ 4

dmcnt = 5

freqMap1

a ⟶ 1
b ⟶ ~~1~~ 2
c ⟶ 1

len = ~~6~~ 5

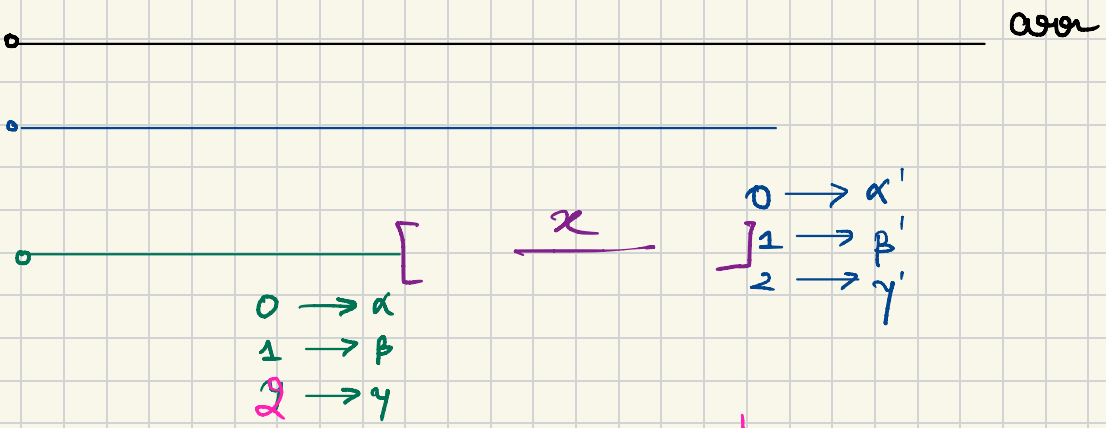
Longest Subarray with equal freq of 0, 1 and 2

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

Brute Force

- Calc. all the subarrays, and store cnt 0, 1 & 2
- store max len

TC : O(N^2)
SC : O(1)



$$\alpha' = \alpha + x$$
$$\beta' = \beta + x$$
$$\gamma' = \gamma + x$$

$$\begin{cases} \beta' - \alpha' = \beta - \alpha \\ \gamma' - \beta' = \gamma - \beta \end{cases}$$

```
int[] arr = { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14
              0, 1, 2, 0, 0, 1, 2, 0, 1, 2, 2, 2, 1, 0, 1 }
```

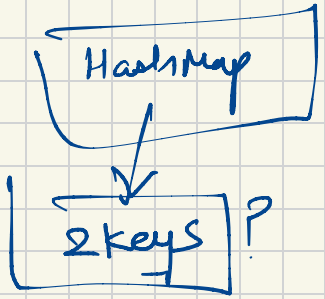
0'	0	1	1	1	2	3	3	3							
1'	0	0	1	1	1	1	2	2							
2'	0	0	0	1	1	1	1	2							
1' - 0'	0	-1	0	0	-1	-2	-1	-1							
2' - 1'	0	0	-1	0	0	0	-1	0							

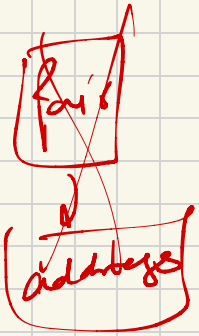
0#0 -1#0

0#-1

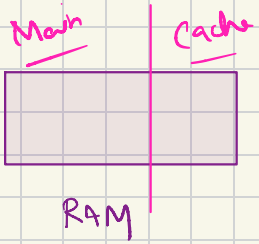
1#0

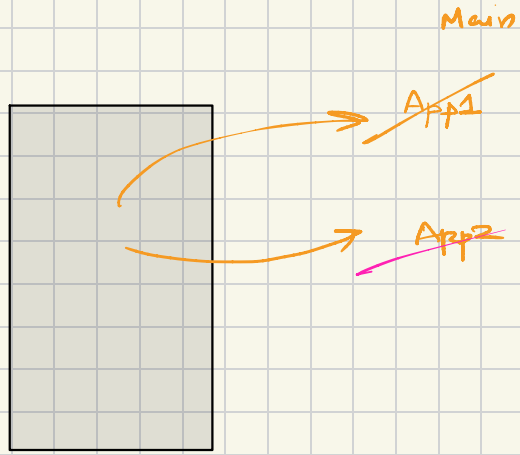
String "key1#key2"

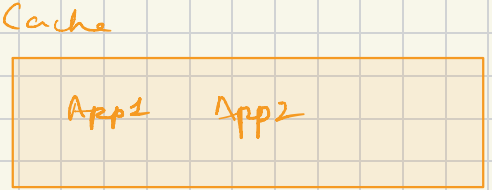




LRU Cache







Cache Memory Management Algorithms

- Least recently used (LRU)
- Least frequently used (LFU)

{Limit = 3 Apps}

App 1 ⟶ Open

App 2 ⟶ Opened

App 3 ⟶ Opened

App 1 ⟶ Popup

App 4 ⟶ Opened

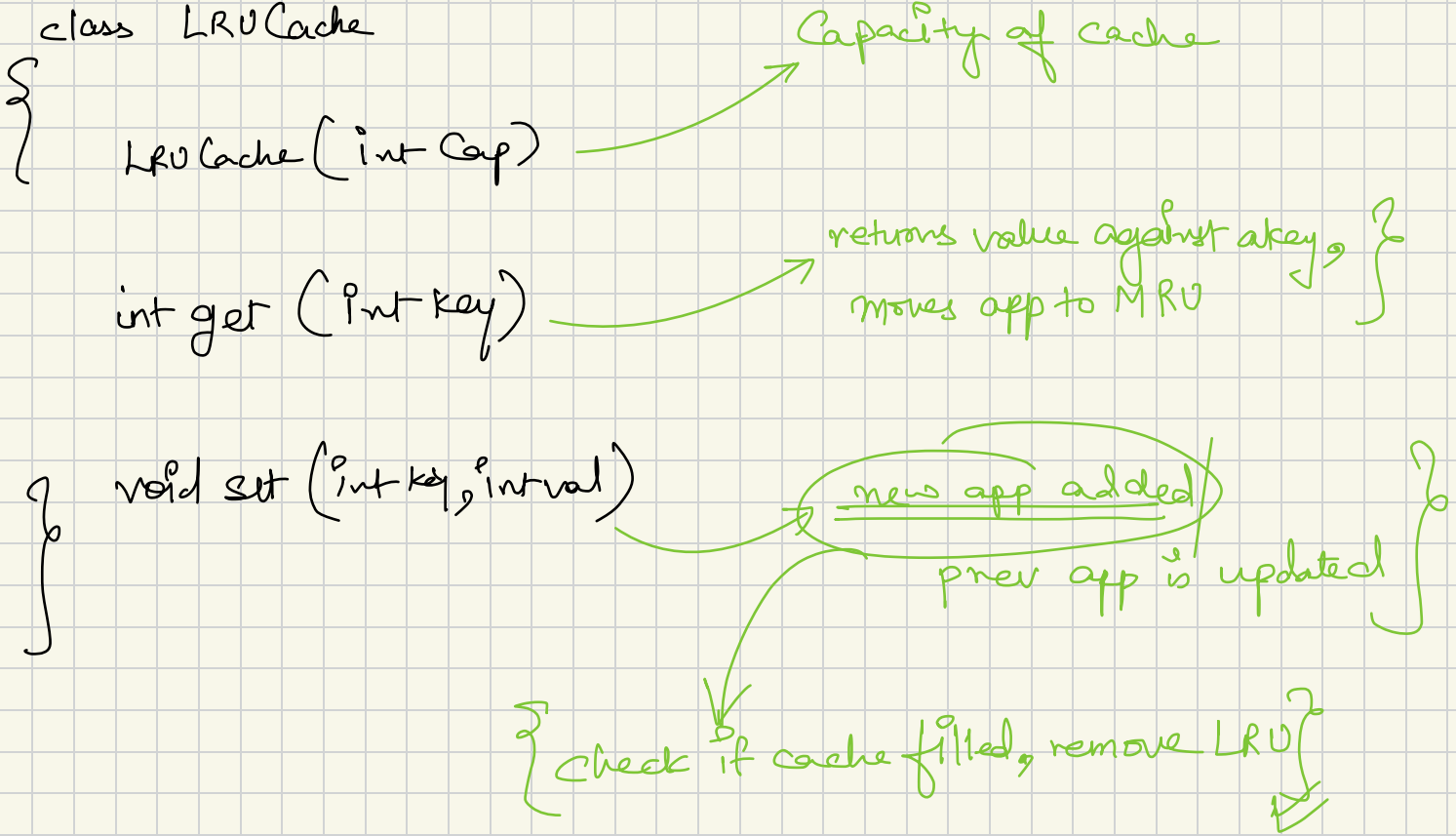
Cache Memory

LRU

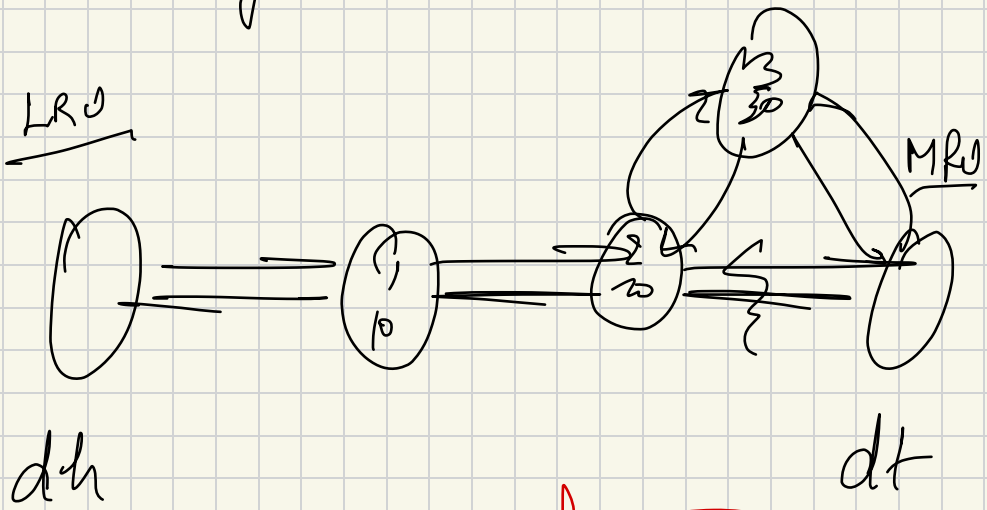
App 1 , ~~App 2~~ , App 3

t=0 ~~t=2~~ t=3

App 4

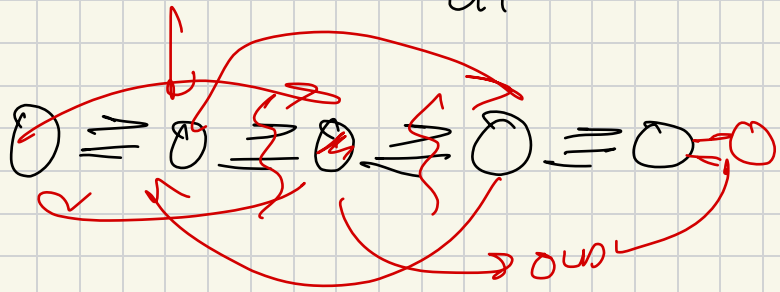

class LRUCache
LRUCache(int cap)
Capacity of cache
int get(int key)
returns value against a key,
moves app to MRU
void set(int key, int val)
new app added
prev app is updated
check if cache filled, remove LRU

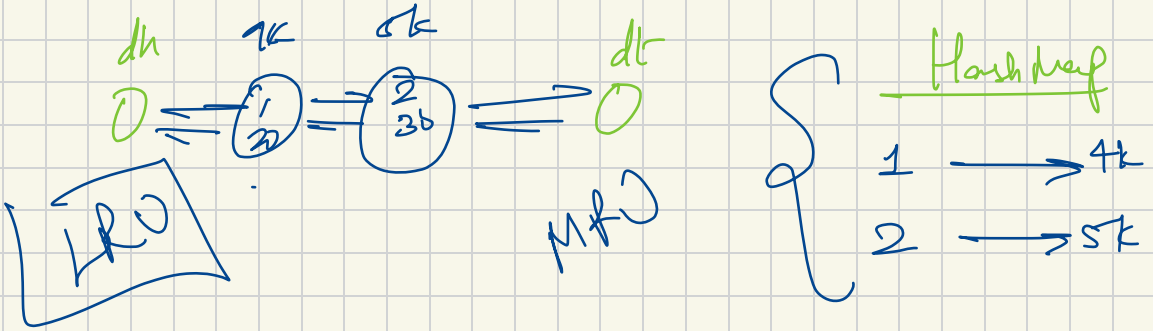
doubly linked list



HashMap

key → address




dh
dt
1
2
3
LRU
MRU
Hash Map
1
2
4k
5k