

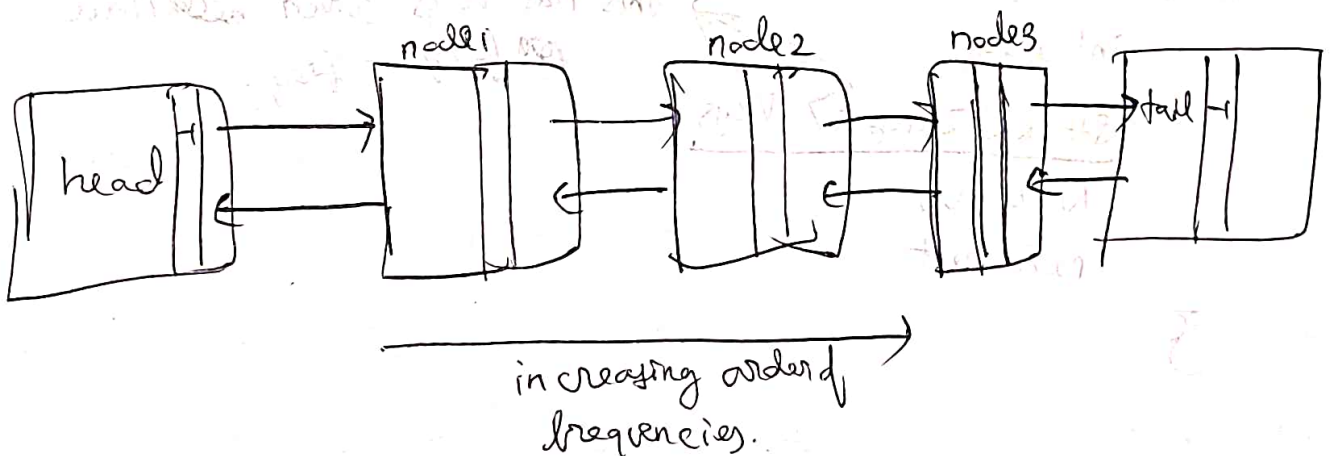
(Max Frequency Stack)

I was not able to do this soln by myself. I will note down, what my thought process was and then I watched NeetCode's explanation and how beautiful the soln is.

(My thought process)

(LFU ; DLL Attempt)

→ Since prior of this I had done a lot of LFU cache problems, hence my mind was biased over using doubly linkedList. And also maxfreq was there, so my mind was mostly diverting towards using DLL for implementing something like maxStack.



I thought like in LFU cache, I could use a LinkedHashSet to maintain order of insertion.

P.T.O.

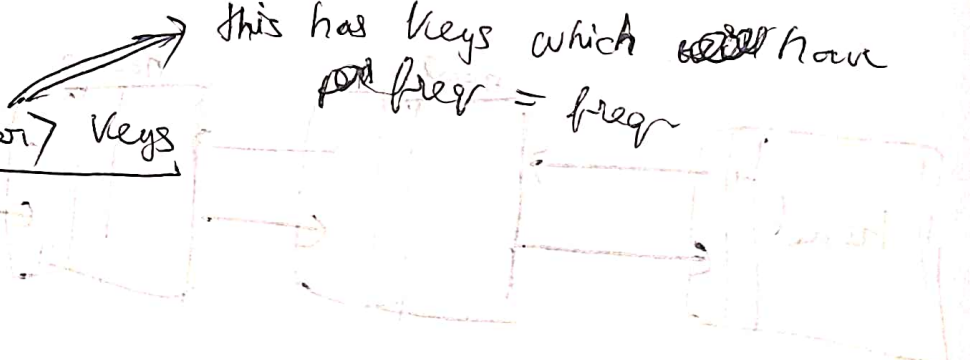
if you are re-reading this and wondering why Linked Hashset,
go ~~back~~ read the LRU cache problem solving experience,
that you had, → { refresh memory
LinkedHashMap, → preserve order of
LinkedHashSet → insertion
So in the condition all removed the
front as least recently used

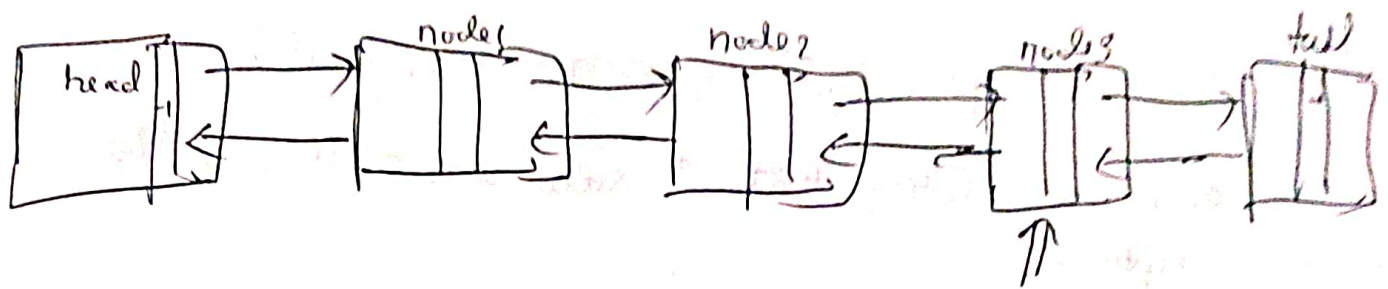
~~But~~ But then I remembered that LinkedHashSet is first element,
will give the lowest in the stack of that frequency, ~~but~~ But
we wanted top element

So modified from (LinkedHashSet → Stack)

```
class Node {  
    int freq;  
    Stack<Integer> Keys  
    Node prev  
    Node next  
}
```

↗ this has keys which ~~are~~ have
~~freq~~ freq = freq

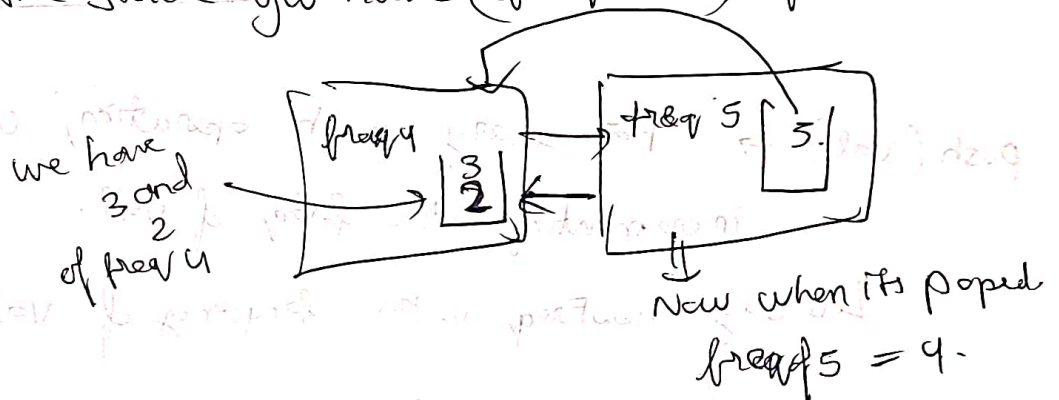




When `pop()` is called just do,
 this nodes `Stack<Integer> - pop.`
 { This is the maxFreq node }

But, But, But there is a problem given,

let's say in the stack you have (~~freq~~ - 5) for val=5.

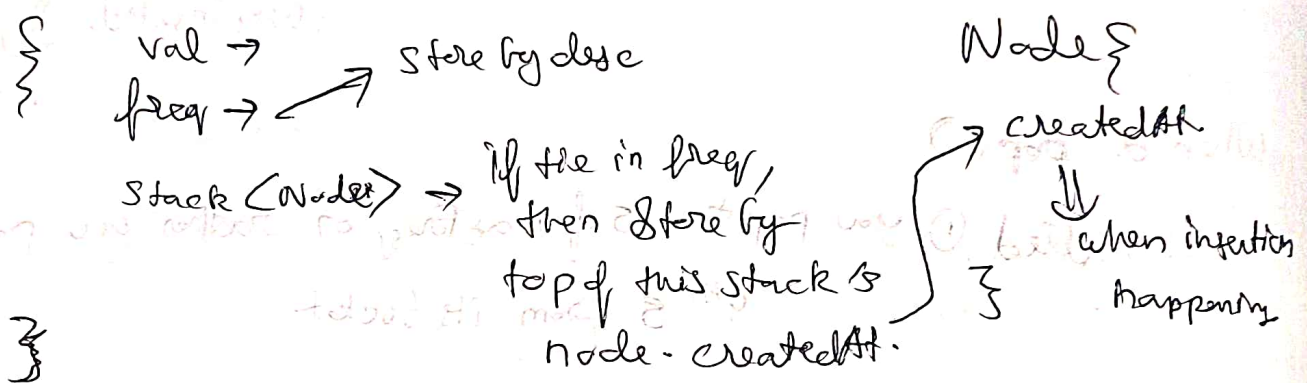


Now you have to move 5 to freq 4.

But the problem is you don't know how the remaining 4, 5's are in order in stack, making it a problem, You are loosing the order and you don't have any way to maintain order of all the 5's, if you keep 5 only in the node, where its maxFreq is.

So, I tried to think more, and thought of using heap,
trying a $O(\log N)$ pop. (Priority Queue Attempt)

when well it will work, I can get the top of the
heap, and I can create a custom DLL stack.
In the max heap I can have an entry as



Well this will support $O(\log N)$ pop.

But what happens in case of push (val).

lets say you are pushing a val, whose freq is not at
top of the priority-queue, so you have to search the
priority-queue manually and reinsert it.

~~It~~ $O(N)$ time, won't work,

(Finally I saw the Editorial)

And the solution, that I saw was so beautifully crafted.

We are using two HashMaps.

1st \rightarrow val \rightarrow Frequency

2nd \rightarrow freq \rightarrow Bucket of keys. \rightarrow [Integer \rightarrow Stack<Integer>]

push(val) \rightarrow For every push operation, we are incrementing the freq of val.

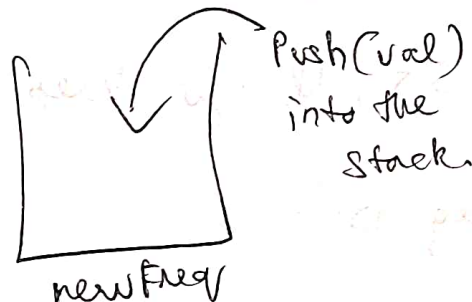
Let say newFreq is the frequency of val

Now in the 2nd HashMap,

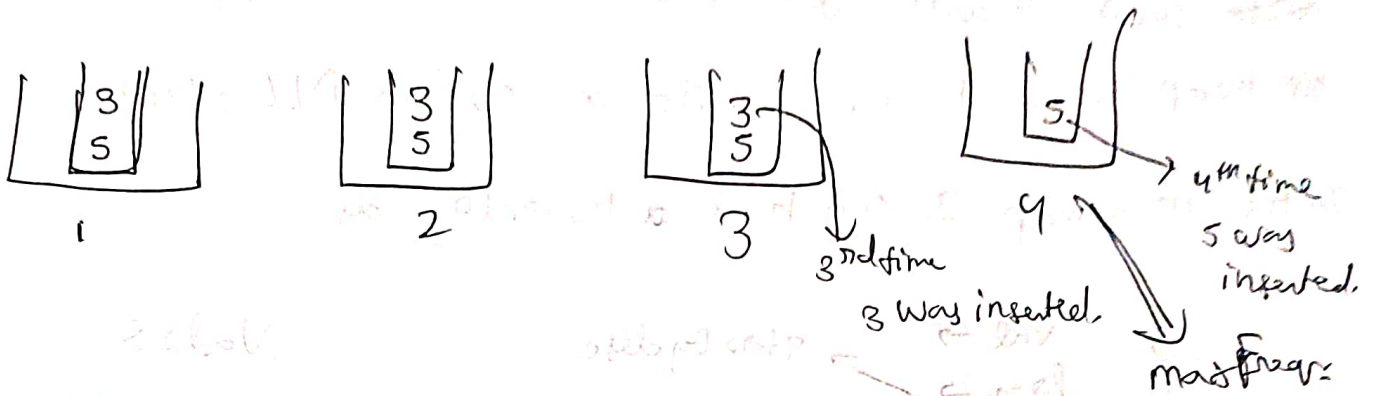
for newFreq insert or push this val to the stack,

Again a push happens, $\text{newFreq} = \text{newFreq} + 1$,

in the newBucket



What technically you are doing is, the i^{th} time any val is getting pushed, you are pushing it into the i^{th} bucket stack of numbers.



When a pop()

is called, ① you pop the 5 of maxFreq, or rather you pop the 4^{th} 5 from its bucket

② Check if 4^{th} Bucket's stack is empty, you do $maxFreq--$, cause you know 5 was inserted 3^{rd} time also so it should be in the stack.

Lastly, keeping a stack for each bucket we are maintain order of insertion into the stack in case of tie,

this is kind of bucket sort only, but such beautiful implementation.

I am amazed.