(All O(1) data structure)

(Prerequisite read explanation of LFU cache)

maxKey() → Since am allowed to return any key with the maxFreq or minFreq in case of a time am using HashSet.

minKey() →

My thought process and the experience I had solving this question for the first time. :

First time, I was trying the first approach of LFU cache

map (String, Integer) → {Key to Freqency Map}

Map (Integer, HashSet<String>) → {Freqency to set of Strings which have that freqency}

Earlier, we were maintaining minFreq, now we have to maintain maxFreq

where does the approach fail?

Since dec is supported,

You do a dec of Key3,

what happens?

removed,

and

you cannot jump to (minFreq=3)

Freq 6 ← Key1 → maxFreq

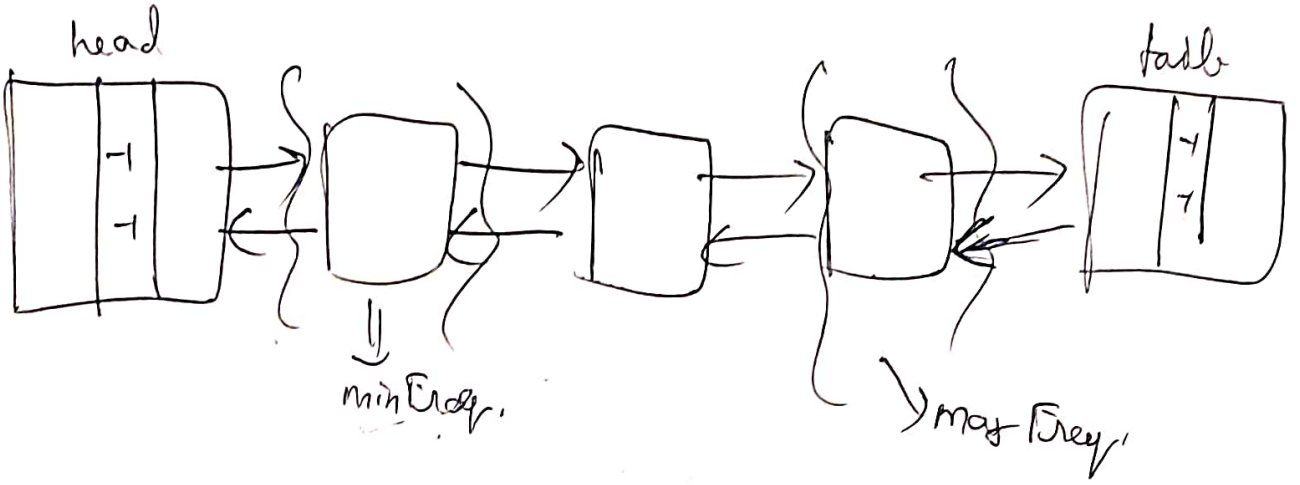
Freq 3 ← Key2 (this should be minFreq now)

(Freq 1 ← Key3) → (minFreq)

So this approach won't work,

New Approach → Prerequisite LFU cache.

Use the doubly LinkedList and maintain HashSet<String> in integer.



Advantage ? → You can directly jump to next minFreq if minFreq is empty or jump to previous maxFreq if maxFreq node becomes empty

incKey() → You maintain a HashMap(Key, Node) → Location in DLL

(Step1) get prevFreq, remove the key from that node

(Step2) (newFreq = prevFreq+1) check if node.next.freq = prevFreq

No? → create a newNode with freq = prevFreq+1 and insert b/w (node and node.next) → update Location in map → put(Key, newNode)

Yes? → insert it in the node → update Location in map → put(Key, newNode)

Similar for decKey() → You get the idea,

Practice writing the code properly.