<https://leetcode.com/problems/maximum-frequency-stack/description/>

Design a stack-like data structure to push elements to the stack and pop the most frequent element from the stack.

Implement the FreqStack class:

* FreqStack() constructs an empty frequency stack.
* void push(int val) pushes an integer val onto the top of the stack.
* int pop() removes and returns the most frequent element in the stack.
* If there is a tie for the most frequent element, the element closest to the stack's top is removed and returned.

**Example 1:**

Input

["FreqStack", "push", "push", "push", "push", "push", "push", "pop", "pop", "pop", "pop"]

[[], [5], [7], [5], [7], [4], [5], [], [], [], []]

Output

[null, null, null, null, null, null, null, 5, 7, 5, 4]

Explanation

FreqStack freqStack = new FreqStack();

freqStack.push(5); // The stack is [5]

freqStack.push(7); // The stack is [5,7]

freqStack.push(5); // The stack is [5,7,5]

freqStack.push(7); // The stack is [5,7,5,7]

freqStack.push(4); // The stack is [5,7,5,7,4]

freqStack.push(5); // The stack is [5,7,5,7,4,5]

freqStack.pop(); // return 5, as 5 is the most frequent. The stack becomes [5,7,5,7,4].

freqStack.pop(); // return 7, as 5 and 7 is the most frequent, but 7 is closest to the top. The stack becomes [5,7,5,4].

freqStack.pop(); // return 5, as 5 is the most frequent. The stack becomes [5,7,4].

freqStack.pop(); // return 4, as 4, 5 and 7 is the most frequent, but 4 is closest to the top. The stack becomes [5,7].

**Constraints:**

* 0 <= val <= 109
* At most 2 \* 104 calls will be made to push and pop.
* It is guaranteed that there will be at least one element in the stack before calling pop.

**Attempt 1: 2023-11-08**

**Solution 1: Priority Queue + Hash Table (60 min)**

class FreqStack {

Map<Integer, Stack<Integer>> map;

Map<Integer, Integer> freq;

int maxFreq;

public FreqStack() {

map = new HashMap<>();

freq = new HashMap<>();

maxFreq = 0;

}

public void push(int val) {

int f = freq.getOrDefault(val, 0) + 1;

if(f > maxFreq) {

maxFreq = f;

}

freq.put(val, f);

if(!map.containsKey(f)) {

map.put(f, new Stack<>());

}

map.get(f).add(val);

}

public int pop() {

int val = map.get(maxFreq).pop();

freq.put(val, maxFreq - 1);

if(map.get(maxFreq).size() == 0) {

maxFreq--;

}

return val;

}

}

/\*\*

\* Your FreqStack object will be instantiated and called as such:

\* FreqStack obj = new FreqStack();

\* obj.push(val);

\* int param\_2 = obj.pop();

\*/

Time Complexity : O(1) FOR PUSH AND POP

Space Complexity : O(N)

**Refer to**

<https://leetcode.com/problems/maximum-frequency-stack/solutions/163410/c-java-python-o-1/>

Hash map freq will count the frequency of elements.

Hash map m is a map of stack.

If element x has n frequency, we will push x n times in m[1], m[2] .. m[n]maxfreq records the maximum frequency.

push(x) will push x tom[++freq[x]]

pop() will pop from the m[maxfreq]

class FreqStack {

HashMap<Integer, Integer> freq = new HashMap<>();

HashMap<Integer, Stack<Integer>> m = new HashMap<>();

int maxfreq = 0;

public void push(int x) {

int f = freq.getOrDefault(x, 0) + 1;

freq.put(x, f);

maxfreq = Math.max(maxfreq, f);

if (!m.containsKey(f)) m.put(f, new Stack<Integer>());

m.get(f).add(x);

}

public int pop() {

int x = m.get(maxfreq).pop();

freq.put(x, maxfreq - 1);

if (m.get(maxfreq).size() == 0) maxfreq--;

return x;

}

}

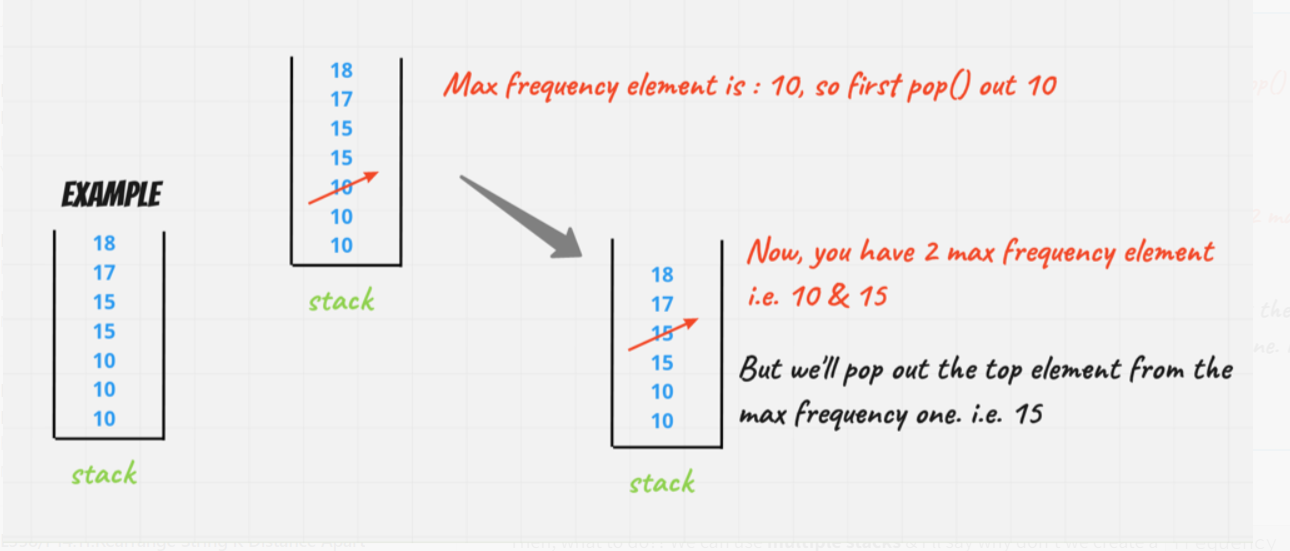
**Refer to**

<https://leetcode.com/problems/maximum-frequency-stack/solutions/1862015/best-explanation-visually/>

Let's understand what we have given is 2 operations of push() & pop().

And what the problem statement is saying `pop` the maximum frequency element from the stack.

Okay, now you know that Stack use a **LIFO** order. So, it'll be a little hard for us to pop() the maximum frequency element out from the stack, if it's in the middle or somewhere. First we have to empty the stack until our element is on the top of the stack & store those element's somewhere.

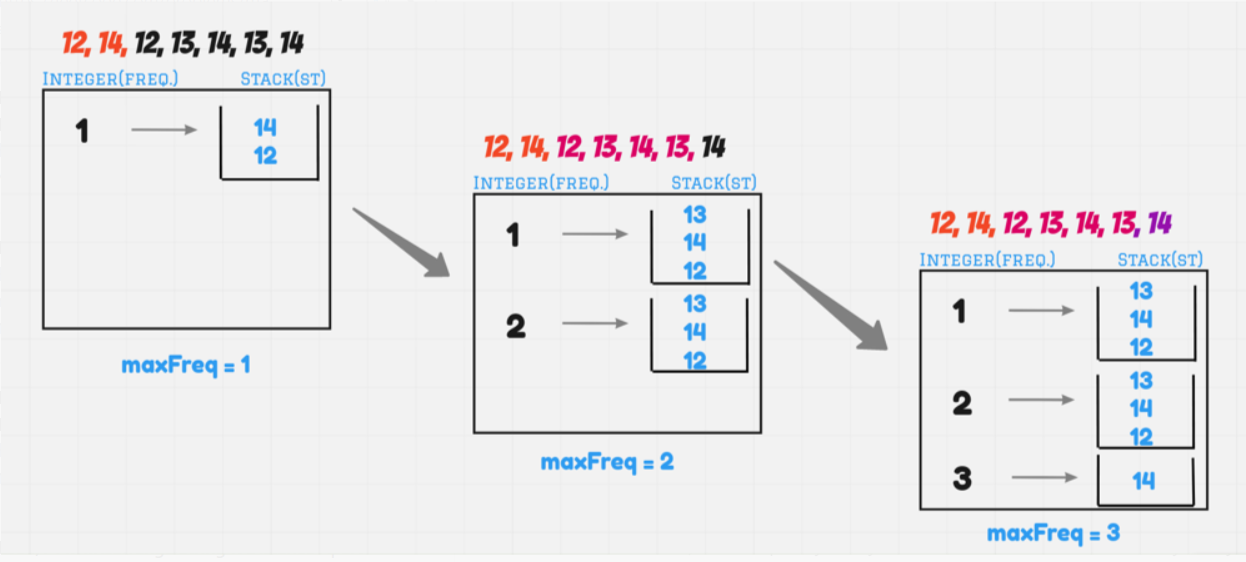


**So, it's not a good approach!!**

Then, what to do?? We can use **multiple stacks** & i'll say why don't we create a frequency stack as well, now you say. What do you mean by that!!**It's sort of a stack like that if same element's re-appear we'll put them into a new stack!**

Well, let's understand with an example:-Let's say we have given these value's

E.g :- 12, 14, 12, 13, 14, 13, 14

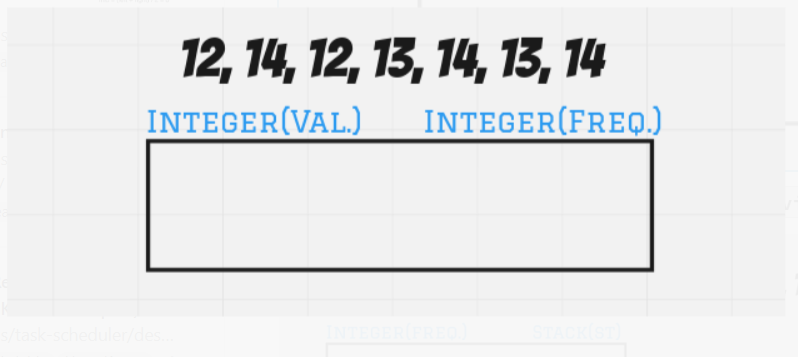


If, you are removing it & stack becomes empty remove it from frequency & reduce the **maxFreq** by 1

But there is **one issue**, for example if you say's let's add **14** into it. Then how can you say that this is **14th** 3rd occurence & put it into a new stack.

Now you'll say, we gonna traverse over the stack & check>> *Smart enough*. But why you are *increasing time complexity!!*

Okay, for this one, we'll create one HashMap of **Integer, Integer** i.e. a frequency map.



Let's understand it back again with same example, but now **visually** :-



Till Now we have understand how push() operation is going on. let's see how pop() operations will look's like



**And Guy's remember when we are poping out store your answer in a variable as at the end, we'll return it.**

Sum Up:-

Craete 2 HashMap & one variable

I hope ladie's n gentlemen approach is absolute clear, **Let's Code it, up**

class FreqStack {

Map<Integer, LinkedList<Integer>> st;

Map<Integer, Integer> map = new HashMap<>();

int maxFreq;

public FreqStack() {

st = new HashMap<>();

map = new HashMap<>();

maxFreq = 0;

}

public void push(int val) {

int currFreq = map.getOrDefault(val, 0);

currFreq++;

map.put(val, currFreq);

if(st.containsKey(currFreq) == false){

st.put(currFreq, new LinkedList<Integer>());

}

st.get(currFreq).addFirst(val);

maxFreq = Math.max(maxFreq, currFreq);

}

public int pop() {

int ans = st.get(maxFreq).removeFirst();

int currFreq = map.get(ans);

currFreq--;

map.put(ans, currFreq);

if(st.get(maxFreq).size() == 0){

maxFreq--;

}

return ans;

}

}