EP. 15

CODING INTERVIEW

TOP INTERVIEW QUESTION

By Ashay Nayak

Source: My Medium Article

By Ashay Nayak

Source: My Medium Article

By Ashay Nayak

Source: My Medium Article

Given an integer array **nums** and an integer **k**, return the **k most frequent elements**. You may return the answer in **any order**.

By Ashay Nayak

EXAMPLES

Example 1:

```
Input: nums = [1,1,1,2,2,3], k = 2
Output: [1,2]
```

Example 2:

```
Input: nums = [1], k = 1
Output: [1]
```

Going to discuss **multiple possible solutions** to solve this problem. Also, I will share **4 problems around k elements** for your practice.

Purpose: Practice of thinking of all possible solutions which help you in interviews.

APPROACH 1

Using Tree Map:

A.) Iterate the given array, calculate and store the frequency of each element using HashMap. Here, the key in HashMap is the element and the value is the frequency of that element in the given array.

B.) TreeMap store data in key-value pair and data is sorted in increasing order of keys. So, iterate through the frequency HashMap and store frequency as key and element as value.

C.) Now, iterate through the TreeMap from the end and print the first k elements, which are the most frequent as data in TreeMap is stored in increasing order of key.

TC → O(nlogn) and SC → O(n)

CODE

Title

```java
class Solution {
    public int[] topKFrequent(int[] nums, int k) {
        int n = nums.length;
        // we are storing the frequency - O(n) time
        Map<Integer,Integer> freq = new HashMap<Integer,Integer>();
        for(int i=0;i<n;i++){
            freq.put(nums[i],freq.getOrDefault(nums[i],0)+1);
        }

        TreeMap<Integer,List<Integer>> map = new TreeMap<>();
        //O(nlogn) time
        for(int key: freq.keySet()){  // iterate through frequency map and store frequency as key in map and add corresponding element as value.
            int val = freq.get(key); // here key is the element and val is the frequency

            if(map.containsKey(val)==false){
                map.put(val,new LinkedList<Integer>());
            }

            map.get(val).add(key);
        }

        List<Integer> ans = new LinkedList<>();
        // O(klogn) time
        while(ans.size()<k){
            Map.Entry<Integer, List<Integer>> entry = map.pollLastEntry(); // doing because tree map is sorted in increasing order and we want k most frequent element so frequency is high at last
            ans.addAll(entry.getValue());
        }

        int result[] = new int[ans.size()]; // it is just we want array to return
        for(int i=0;i<ans.size();i++){
            result[i]=ans.get(i);
        }

        return result;
    }
}
```

APPROACH 2

Using Min Heap:

A.) Iterate the given array, calculate and store the frequency of each element using hashmap.

B.) Create Min Heap of size k. Why Min-Heap? Let's consider you have nums=[3, 4, 1, 5, 5] and you need to return two(k=2) largest elements. Here, min-heap can help. How? First, Create a min heap mh. Now, iterate through nums and add elements in mh. Because we need the two largest elements, we will restrict the size of mh to two. Now, start adding 3 then 4 then 1 (mh has (3, 4, 1)). After adding 1, the size of mh becomes greater than two, so we need to remove an element and it will remove 1 as it is a min heap(now, mh has (3, 4)). Keep iterating and adding element. As soon as it crosses size 2, remove element from heap. We will end up with 5 and 5 in min heap which are our largest 2 elements. Same logic is applied here. See code 👉

TC → O(nlogk) and SC → O(n)

CODE

Title

```java
class Solution{
    public int[] topKFrequent(int[] nums, int k){
        int n = nums.length;
        // we are storing the frequency - O(n) time
        Map<Integer,Integer> freq = new HashMap<Integer,Integer>();
        for(int i=0;i<n;i++){
            freq.put(nums[i],freq.getOrDefault(nums[i],0)+1);
        }

        PriorityQueue<Map.Entry<Integer,Integer>> minHeap = new PriorityQueue<>((a,b)->(a.getValue()-b.getValue())); // we are using minHeap based on frequency

        //O(nlogk) time
        for(Map.Entry<Integer,Integer> entry:freq.entrySet()){
            minHeap.add(entry);
            if(minHeap.size()>k)minHeap.poll(); // keeping min Heap size to k only
        }
     /* consider that 3,4,1,5 are my elements and we want largest two elements only, then min heap can help here. run through the elements using above code.
add 3 then 4 then 1, now when you add 1, size is greater than 2 so we need to remove element and it will remove 1 as it is min heap.
Now add 5 and again size is greater than 2, we need to remove and this time it will remove 3 as it is min heap so we end up getting largest 2 elements.*/

        List<Integer> ans = new LinkedList<>();
        //O(klogk) time
        while(ans.size()<k){
            Map.Entry<Integer, Integer> entry = minHeap.poll();
            ans.add(entry.getKey());
        }

        int result[] = new int[ans.size()]; // it is just we want array to return
        for(int i=0;i<ans.size();i++){
            result[i]=ans.get(i);
        }

        return result;

    }
}
```

LEETCODE PRACTICE QUESTIONS

FYI, MinHeap or MaxHeap Technique is generally used for Kth problems like smallest K elements. See some Leetcode questions for practice:

215. Kth Largest Element in an Array
347. Top K Frequent Elements
973. K Closest Points to Origin
230. Kth Smallest Element in a BST

APPROACH 3

Using Bucket – Best Solution:

A.) Iterate the given array, calculate and store the frequency of each element using HashMap.

First step is common for all approaches. To understand the bucket method,

check out my article link given in the comment below.

Hint: Bucket is an array of linkedList.

TC → O(n) and SC → O(n)

ALSO, CHECKOUT MY OTHER ARTICLES ON JAVA - INTERVIEW PREPARATION (LINK IN COMMENT)

By Ashay Nayak

Source: My Medium Article

TO BE CONTINUED...

By Ashay Nayak

Source: My Medium Article