Top 'K' Frequent Numbers (medium)

We'll cover the following

Problem Statement

Given an unsorted array of numbers, find the top 'K' frequently occurring numbers in it.

Example 1:

```
Input: [1, 3, 5, 12, 11, 12, 11], K = 2
Output: [12, 11]
Explanation: Both '11' and '12' apeared twice.
```

Example 2:

```
Input: [5, 12, 11, 3, 11], K = 2
Output: [11, 5] or [11, 12] or [11, 3]
Explanation: Only '11' appeared twice, all other numbers appeared once.
```

Try it yourself

Try solving this question here:

Java | Python3 | JS | C++

```
const find_k_frequent_numbers = function(nums, k) {
  topNumbers = [];
  // TODO: Write your code here
  return topNumbers;
};


console.log(`Here are the K frequent numbers: ${find_k_frequent_numbers([1, 3, 5, 12, 11, 12, 11], 2)}`)
console.log(`Here are the K frequent numbers: ${find_k_frequent_numbers([5, 12, 11, 3, 11], 2)}`)

```

RUN SAVE RESET

Solution

This problem follows Top 'K' Numbers. The only difference is that in this problem, we need to find the most frequently occurring number compared to finding the largest numbers.

We can follow the same approach as discussed in the **Top K Elements** problem. However, in this problem, we first need to know the frequency of each number, for which we can use a **HashMap**. Once we have the frequency map, we can use a **Min Heap** to find the 'K' most frequently occurring number. In the **Min Heap**, instead of comparing numbers we will compare their frequencies in order to get frequently occurring numbers

Code

Here is what our algorithm will look like:

Java | Python3 | C++ | JS

```
const Heap = require('./collections/heap'); //http://www.collectionsjs.com


function find_k_frequent_numbers(nums, k) {
  // find the frequency of each number
  numFrequencyMap = {};
  nums.forEach((num) => {
    if (!(num in numFrequencyMap)) {
      numFrequencyMap[num] = 1;
    } else {
      numFrequencyMap[num]++;
    }
  });

  const minHeap = new Heap([], null, ((a, b) => b[0] - a[0]));

  // go through all numbers of the numFrequencyMap and push them in the minHeap, which will have
  // top k frequent numbers. If the heap size is more than k, we remove the smallest(top) number
  Object.keys(numFrequencyMap).forEach((num) => {
    minHeap.push([numFrequencyMap[num], num]);
```

```
21    if (minHeap.length > k) {
22      minHeap.pop();
23    }
24  });
25
26  // create a list of top k numbers
27  const topNumbers = [];
28  while (minHeap.length > 0) {
```

RUN SAVE RESET

Close

Output 7.636s

```
Here are the K frequent numbers: 12,11
Here are the K frequent numbers: 12,11
```

Time complexity

The time complexity of the above algorithm is $O(N + N * logK)$.

Space complexity

The space complexity will be $O(N)$. Even though we are storing only 'K' numbers in the heap. For the frequency map, however, we need to store all the 'N' numbers.

Back
Connect Ropes (easy)

Next
Frequency Sort (medium)

Mark as Completed

Report an Issue Ask a Question