Top K Frequent Elements

Top K Frequent Elements

Medium

Given an integer array `nums` and an integer `k`, return the `k` most frequent elements within the array.

The test cases are generated such that the answer is always unique.

You may return the output in any order.

Example 1:

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

Output: [2,3]
```

Copy

Example 2:

```
Input: nums = [7,7], k = 1

Output: [7]
```

Copy

Constraints:

`1 <= nums.length <= 10^4`. `-1000 <= nums[i] <= 10001 <= k <= number of distinct elements in nums`.

- `1 <= nums.length <= 10^4`.
- `1000 <= nums[i] <= 1000`
- `1 <= k <= number of distinct elements in nums`.

My Initial thought :

Implementing bucket sorting and return any frequent buckets that have an element > 1 and return the elements of the frequent buckets

I did not figure it out in the end, here is the solution I had help on.

```
from collections import Counter
from typing import List

class Solution:
    def topKFrequent(self, nums: List[int], k: int) -> List[int]
        # Step 1: Scatter - Count frequencies
        frequency = Counter(nums)  # Map each element to its fre
        max_frequency = max(frequency.values())  # Find the high

        # Create buckets for frequencies (Scatter into buckets)
        buckets = [[] for _ in range(max_frequency + 1)]
        for num, freq in frequency.items():
            buckets[freq].append(num)  # Place numbers into the

        # Step 2: Sort - Sort each bucket (trivial here since nu
        # Note: We don't sort within a bucket, as sorting by fre

        # Step 3: Gather - Collect results from buckets in rever
        result = []
        for freq in range(len(buckets) - 1, 0, -1):  # Traverse
            for num in buckets[freq]:
                result.append(num)  # Gather elements from the c
                if len(result) == k:  # Stop when we have k elem
                    return result
```

```
frequency = Counter(nums)
```

A dictionary is created to map the frequency of each number

```
  max_frequency = max(frequency.values())  # Find the highest fr
```

which we then find the highest frequency

```
nums = [1, 1, 1, 2, 2, 3]
Frequency:
{1: 3, 2: 2, 3: 1}
```

Bucket Sort - Scatter - sort - merge

Scattering the elements into individual buckets (in this case its by their frequency)

```
frequency = Counter(nums)
buckets = [[] for _ in range(max_frequency + 1)] #Remember that
for num, freq in frequency.items():
    buckets[freq].append(num)
```

A mapping of numbers to their frequencies is used as a mapping method

So we call out the frequency.items (bring the key-value pairs to focus)

and for every number, we map it to a bucket indexed by its frequency

```
for freq in range(len(buckets) - 1, 0, -1):
    for num in buckets[freq]:
        result.append(num)
```

```
            if len(result) == k:
                return result
```

Starting from the highest frequency, we traverse down and append the number onto the result

Conceptual Questions :

Why do we use `buckets = [[] for _ in range(max_frequency + 1)]` instead of just `buckets = []`?
We initialise the bucket size to accommodate the frequency of the number

What does the `max_frequency + 1` ensure when creating the list of buckets?
Lists are inclusve, so we add an extra +1 to include the 0 frequency numbers

The mapping function is decided by the frequency of the characters, so we sort it to a bucket depending on the frequency

Top K frequent elements wants you to return the elements that matches K groups of elements, K elements is interchangeable, a generic bucket sort sorts an array of integers from a specified order.

We don't explicitly sort elements inside each bucket as we aren't looking to sort the numbers in any way, we only require the frequency of each elements

Reverse order so that we take the highest frequency elements to lowest until K is met

By checking length of result everyime a number is appended from the buckets[frequency]