
Problem Statement:

Find the top k most frequent elements from a list of integers.

Blind Solution

```python
def most_frequent_elements_blind(nums: List[int], element_count: int) -> List[int]:
    freq_counts = defaultdict(int) # A default dict initialises new keys without key errors
    most_frequent_keys = []

    for num in nums:
        freq_counts[num] += 1

    print(freq_counts)

    for _ in range(element_count):
        max_key = max(freq_counts, key=freq_counts.get)
        most_frequent_keys.append(max_key)
        freq_counts.pop(max_key)

    return most_frequent_keys
```

Step-by-Step Breakdown

1. **Input:**
    - nums = [1, 2, 2, 3, 3, 3, 4, 4, 4, 4, 4]
    - element_count = 2.
2. **Frequency Count:**
   Use a defaultdict to count the occurrences of each number. After iterating through nums, the frequency map will look like this:
   freq_counts = { 1: 1,

   2: 2,

   3: 3,

   4: 5 }
3. **Finding Maximums Repeatedly:**
   For each of the top k elements (element_count = 2):
    - **First Iteration:**
        - max_key = 4 (highest frequency is 5).
        - Append 4 to most_frequent_keys.
        - Pop 4 from the frequency map:
        - freq_counts = { 1: 1,

          2: 2,

3: 3 }

- **Second Iteration:**
  - max_key = 3 (highest frequency is now 3).
  - Append 3 to most_frequent_keys.
  - Pop 3 from the frequency map:
  - freq_counts = { 1: 1,

    2: 2 }

4. **Output:** most_frequent_keys = [4, 3]
5. **Efficiency:**
   - Counting frequencies: O(n), where n is the number of elements in nums.
   - Finding the max key k times: O(k·m), where m is the size of freq_counts.
   - Total: **O(k·m+n)**

Optimised Heap-Based Solution

```
def most_frequent_elements(nums: List[int], element_count: int) -> List[int]:
    # Count freq of each element
    freq_counts = Counter(nums) # O(n)

    # Use a heap to find top k elements
    most_frequent_keys = heapq.nlargest(element_count, freq_counts.keys(), key=freq_counts.get)

    return most_frequent_keys
```

Step-by-Step Breakdown

1. **Input:**
   - nums = [1, 2, 2, 3, 3, 3, 4, 4, 4, 4, 4]
   - element_count = 2.
2. **Frequency Count:**
   Use Counter to count occurrences of each number. After processing nums, the frequency map is:
   - freq_counts = { 4: 5,

     3: 3,

     2: 2,

     1: 1})
3. **Using heapq.nlargest:**

- heapq.nlargest() finds the top k elements efficiently by building a max-heap under the hood.
- The heap uses the freq_counts.get function to prioritise elements by their frequency.
- Steps:
  - Extract 4 (highest frequency).
  - Extract 3 (second highest frequency).

4. **Output:**
   - most_frequent_keys = [4, 3]
5. **Efficiency:**
   - Counting frequencies: O(n), where n is the number of elements in nums.
   - Building the heap and extracting the top k elements: O(n+k·logn).
   - Total: **O(n+k·logn)**

Comparison

Aspect	Blind Solution	Heap Solution
Efficiency	O(k·m+n)	O(n+k·logn)
Memory Usage	Minimal (uses defaultdict)	Moderate (heap + Counter)
Ease of Implementation	Simple and intuitive	Slightly more complex
Scalability	Slower for large inputs	Scales better for large inputs

---