Top K Frequent Words

Given a non-empty list of words, return the k most frequent elements.

Your answer should be sorted by frequency from highest to lowest. If two words have the same frequency, then the word with the lower alphabetical order comes first.

Example 1:

```
Input: ["i", "love", "leetcode", "i", "love", "coding"], k = 2
Output: ["i", "love"]
Explanation: "i" and "love" are the two most frequent words.
    Note that "i" comes before "love" due to a lower alphabetical order.
```

Example 2:

```
Input: ["the", "day", "is", "sunny", "the", "the", "the", "sunny", "is", "is"], k = 4
Output: ["the", "is", "sunny", "day"]
Explanation: "the", "is", "sunny" and "day" are the four most frequent words,
    with the number of occurrence being 4, 3, 2 and 1 respectively.
```

Note:

1. You may assume k is always valid, $1 \leq k \leq$ number of unique elements.
2. Input words contain only lowercase letters.

Follow up:

1. Try to solve it in $O(n \log k)$ time and $O(n)$ extra space.
2. Can you solve it in $O(n)$ time with only $O(k)$ extra space?

Solution 1

```
class Solution {
public List<String> topKFrequent(String[] words, int k) {
HashMap<String, Integer> map = new HashMap();
for(String s : words){
map.put(s, map.getOrDefault(s, 0) + 1);
}
PriorityQueue<Map.Entry<String, Integer>> pq = new PriorityQueue<>((a, b) ->
a.getValue() == b.getValue() ? b.getKey().compareTo(a.getKey())
: a.getValue() - b.getValue());
for(Map.Entry<String, Integer> m : map.entrySet()){
if(pq.size() < k){
pq.offer(m);
}else{
if(pq.peek().getValue() < m.getValue()){
pq.poll();
pq.offer(m);
}else if(pq.peek().getValue() == m.getValue() &&
pq.peek().getKey().compareTo(m.getKey()) > 0){
pq.poll();
pq.offer(m);
}
}
}
List<String> rs = new ArrayList<>();
while(!pq.isEmpty()) rs.add(0, pq.poll().getKey());
return rs;
}
}
```

written by ngoc_lam original link here

Solution 2

```
       public List<String> topKFrequent(String[] words, int k) {
            Map<String, Integer> map = new HashMap<>();
            for (String word : words) {
                map.put(word, map.getOrDefault(word, 0) + 1);
            }

            SortedSet<Map.Entry<String, Integer>> sortedset = new TreeSet<>(
                    (e1, e2) -> {
                        if (e1.getValue() != e2.getValue()) {
                            return e2.getValue() - e1.getValue();
                        } else {
                            return e1.getKey().compareToIgnoreCase(e2.getKey());
                        }
                    });
            sortedset.addAll(map.entrySet());

            List<String> result = new ArrayList<>();
            Iterator<Map.Entry<String, Integer>> iterator = sortedset.iterator();
            while (iterator.hasNext() && k-- > 0) {
                result.add(iterator.next().getKey());
            }
            return result;
        }
```

written by fishercoder original link here

Solution 3

This problem is quite similar to the problem Top K Frequent Elements. You can refer to this post for the solution of the problem.

We can solve this problem with the similar idea:
Firstly, we need to calculate the frequency of each word and store the result in a hashmap.

Secondly, we will use bucket sort to store words. Why? Because the minimum frequency is greater than or equal to 1 and the maximum frequency is less than or equal to the length of the input string array.

Thirdly, we can define a trie within each bucket to store all the words with the same frequency. With Trie, it ensures that the lower alphabetical word will be met first, saving the trouble to sort the words within the bucket.

From the above analysis, we can see the time complexity is O(n).
Here is my code:

```
public List<String> topKFrequent(String[] words, int k) {
        // calculate frequency of each word
        Map<String, Integer> freqMap = new HashMap<>();
        for(String word : words) {
            freqMap.put(word, freqMap.getOrDefault(word, 0) + 1);
        }
        // build the buckets
        TrieNode[] count = new TrieNode[words.length + 1];
        for(String word : freqMap.keySet()) {
            int freq = freqMap.get(word);
            if(count[freq] == null) {
                count[freq] = new TrieNode();
            }
            addWord(count[freq], word);
        }
        // get k frequent words
        List<String> list = new LinkedList<>();
        for(int f = count.length - 1; f >= 1 && list.size() < k; f--) {
            if(count[f] == null) continue;
            getWords(count[f], list, k);
        }
        return list;
    }

    private void getWords(TrieNode node, List<String> list, int k) {
        if(node == null) return;
        if(node.word != null) {
            list.add(node.word);
        }
        if(list.size() == k) return;
        for(int i = 0; i < 26; i++) {
            if(node.next[i] != null) {
                getWords(node.next[i], list, k);
            }
        }
    }
```

```
    private boolean addWord(TrieNode root, String word) {
        TrieNode curr = root;
        for(char c : word.toCharArray()) {
            if(curr.next[c - 'a'] == null) {
                curr.next[c - 'a'] = new TrieNode();
            }
            curr = curr.next[c - 'a'];
        }
        curr.word = word;
        return true;
    }

    class TrieNode {
        TrieNode[] next;
        String word;
        TrieNode() {
            this.next = new TrieNode[26];
            this.word = null;
        }
    }
```

written by yro original link here