堆+字典树：前K个高频单词

中等/最小堆、堆排序、桶排序、trie树

学习目标

- 掌握Map的使用方式
- 掌握排序的复杂度
- 掌握priorityQueue数据结构的应用

题目描述

给一非空的单词列表，返回前 *k* 个出现次数最多的单词。

返回的答案应该按单词出现频率由高到低排序。如果不同的单词有相同出现频率，按字母顺序排序。

输入: ["i", "love", "leetcode", "i", "love", "coding"], k = 2
输出: ["i", "love"]
解析: "i" 和 "love" 为出现次数最多的两个单词，均为2次。
注意，按字母顺序 "i" 在 "love" 之前。

输入: ["the", "day", "is", "sunny", "the", "the", "the", "sunny", "is", "is"], k = 4
输出: ["the", "is", "sunny", "day"]
解析: "the", "is", "sunny" 和 "day" 是出现次数最多的四个单词，
出现次数依次为 4, 3, 2 和 1 次。

一. Comprehend 理解题意

该题可以理解为先统计每个单词出现的频率，然后按频率进行排序的问题。

如输入[“i”, “love”, “leetcode”, “i”, “love”, “coding”] k = 2

1、统计每个单词出现频率

单词	频率
i	2
love	2
leetcode	1
coding	1

2、按频率进行排序

单词	频率
i	2
love	2
leetcode	1
coding	1

3、截取出现频率最高的前k个单词

单词	频率
i	2
love	2

二. Choose 数据结构及算法思维选择

方案一：简单排序（暴力解法）

- 统计每个单词出现频率，按频率从高到低排序
- 数据结构：字典
- 算法思维：排序

方案二：优先队列（优化解法）

- 统计每个单词出现频率，维护一个最小堆，存储出现频率最高的k个单词。
- 数据结构：字典，最小堆
- 算法思维：排序

二. Choose 数据结构及算法思维选择

数据结构选择：字典（map）
作用：存储每个单词出现频率

Map是一组键值对的结构，具有极快的查找速度。

如：假设要根据同学的名字查找对应的成绩，如果用Array实现，需要两个Array：

```
names = [‘Alice', ‘Jack', ‘Bob’];
scores = [80, 85, 90];
```

给定一个名字，要查找对应的成绩，就先要在names中找到对应的位置，再从scores取出对应的成绩，Array越长，耗时越长。如果用Map实现，只需要一个“名字”-“成绩”的对照表，直接根据名字查找成绩，无论这个表有多大，查找速度都不会变慢。

用Java写一个Map如下：

```
Map<String, Integer> map = new LinkedHashMap<String, Integer>() {{put(”Alice", 80);put(”Jack",85);}};
map.get(Alice")
```

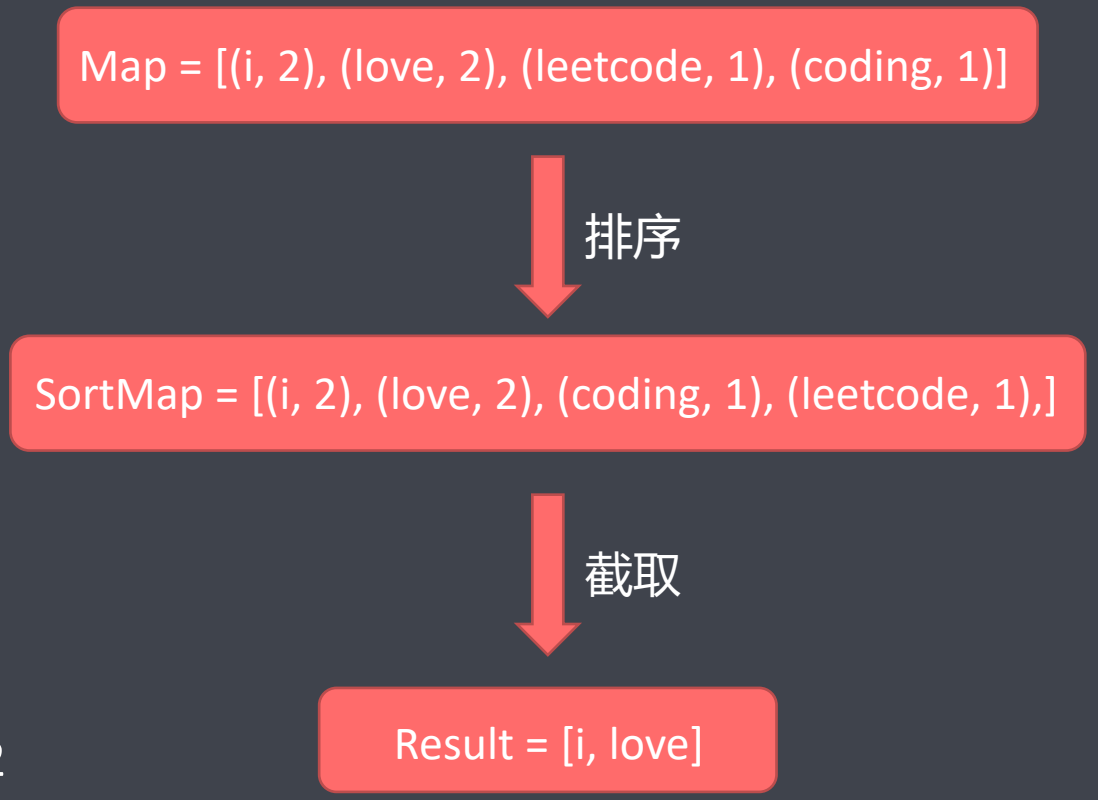
输出： 85

三. Code 基本解法及编码实现

解法一：简单排序（暴力解法）

1. 使用Map存储每个单词出现的频率
2. 按照频率从高到低进行排序。（如果出现频率相同，按照字母顺序排序）
3. 截取出现频率最高的k个字典元素，只取key。

如输入[“i”, “love”, “leetcode”, “i”, “love”, “coding”] k = 2



三. Code 基本解法及编码实现

解法一：简单排序思考

1、如果k远小于数组长度，全部排序是否造成时间浪费？

2、当出现频率相同时如何保证按字母顺序排序？

三. Code 基本解法及编码实现

解法一：暴力解法复杂度分析

时间复杂度：O(nlogn)

- 遍历字符串数组统计单词频率O(n)
- 排序O(nlogn)
- O(n) +O(nlogn)
- 综合时间O(nlogn)

空间复杂度：O(n)

- 存储每个单词出现频率O(n)
- 存放答案O(n)
- O(n) +O(n)=O(2n)
- 忽略常数后：O(n)

三. Code 基本解法及编码实现

解法一：简单排序参考代码

Java编码实现

```
class Solution {
    public List<String> topKFrequent(String[] words, int k) {
        //初始化一个map，用于存放每个单词出现频率
        Map<String, Integer> countFreq = new HashMap();
        //遍历整个字符串数组，如果某个单词出现，频率加1
        for (String word: words) {
            countFreq.put(word, countFreq.getOrDefault(word, 0) + 1);
        }
        //初始化字符串数组用于存储排序后的单词
        List<String> result = new ArrayList(countFreq.keySet());
        //按照频率排序，如果频率相同，则选择字母顺序靠前的单词，否则选择出现频率高的单词
        Collections.sort(result, (word1, word2) -
> countFreq.get(word1).equals(countFreq.get(w2)) ?
                word1.compareTo(word2) : countFreq.get(word2) - countFreq.get(word1));
        //返回排序后前k个单词
        return result.subList(0, k);
    }
}
```

执行结果： 通过 显示详情 >

执行用时： 9 ms ，在所有 Java 提交中击败了 46.14% 的用户

内存消耗： 39.1 MB ，在所有 Java 提交中击败了 41.32% 的用户

三. Code 基本解法及编码实现

解法二：优化解法 用最小堆优化排序

- 由于解法一需要对所有单词的频率进行排序，即便使用快排，时间复杂度也不可能低于O(nlogn)。但是结果只需要输出top k个元组，只需要知道排名前k个单词，不需要知道k+1之后的排名，可以节省排序时间
- 维护一个最小堆可以只保留频率排名前k个单词，时间复杂度只有O(logk)

三. Code 基本解法及编码实现

解法二：优化解法 用最小堆优化排序

1. 使用Map存储每个单词出现的频率
2. 然后将其添加到大小为 k 的最小堆中。它将频率最小的候选项放在堆的顶部。如果新元素的频率高于堆顶元素，则入堆。
3. 最后，我们从堆中弹出最多 k 次，并反转结果，就可以得到前 k 个高频单词。截取出现频率最高的k个字典元素，只取key。

优点：不用全部排序

三. Code 基本解法及编码实现

解法二：最小堆解法复杂度分析

时间复杂度：O(nlogk)

- 遍历单词数据统计频率O(n)
- 建立最小堆O(logk)，对每个单词都要执行一遍插入和删除操作，时间O(nlogk)
- 因为 k ≤n 的值，总时间 O(nlogk)

空间复杂度：O(n)

- 存储每个单词出现频率O(n)
- 存放答案O(k)
- O(n) +O(k)=O(n)

三. Code 基本解法及编码实现

解法二：最小堆解法参考代码

Java编码实现

```java
class Solution {
    public List<String> topKFrequent(String[] words, int k) {
        //初始化一个map，用于存放每个单词的出现频率
        Map<String, Integer> countFreq = new HashMap();
        //遍历整个字符串数组，如果某个单词出现，频率加1
        for (String word: words) {
            countFreq.put(word, countFreq.getOrDefault(word, 0) + 1);
        }
        //初始化一个优先级队列，排序方式为优先按频率从高到低排序，如果频率相同，则按字母顺序
        PriorityQueue<String> heapSort = new PriorityQueue<String>(
                (word1, word2) -> countFreq.get(word1).equals(countFreq.get(word2)) ?
                word2.compareTo(word1) : countFreq.get(word1) - countFreq.get(word2) );
        //对于每一个单词，入队，如果队列大小超过k，则弹出队首，维持队列大小为k
        for (String word: countFreq.keySet()) {
            heapSort.offer(word);
            if (heapSort.size() > k) heapSort.poll();
        }
        //最后队列中存储的k个单词为出现频率最高的单词，逆序输出
        List<String> ans = new ArrayList();
        while (!heapSort.isEmpty()) ans.add(heapSort.poll());
        Collections.reverse(ans);
        return ans;
    }
}
```

执行结果： 通过 显示详情 >

执行用时： 8 ms ，在所有 Java 提交中击败了 87.73% 的用户

内存消耗： 39.3 MB ，在所有 Java 提交中击败了 11.29% 的用户

四. Consider 思考更优解

1. 排序优化

- 单词数组长度有限，考虑此特点，是否有比最小堆更简单的排序方式？

2. 同频率单词选择优化

- 若相同频率出现多个单词，如何优化单词之间按照字母顺序进行排序？

五. Code 最优解思路及编码实现

解法三：优化解法 桶排序+trie树

1. 统计每个单词的频率并将结果存储在map中。
2. 使用桶排序来存储单词频率，相同频率的单词放在同一个桶中。

为什么？

因为最小频率大于或等于1，最大频率小于或等于输入字符串数组的长度。

3. 在每个存储桶中定义一个trie树来存储所有具有相同频率的单词。使用trie，可以确保首先满足排序靠前的字母优先选择，从而省去了对存储桶中的单词进行排序的麻烦。

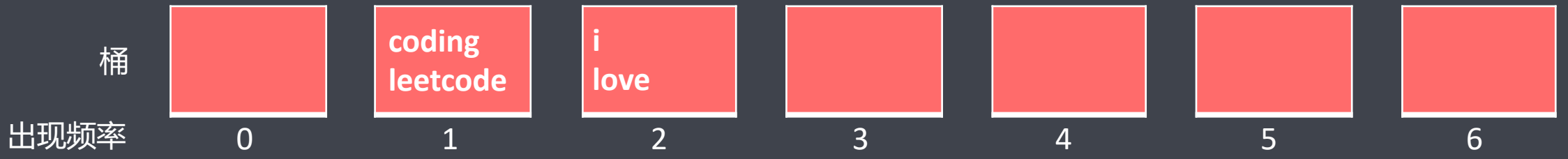
✓ 通过以上分析，我们可以看到时间复杂度为O（n）。

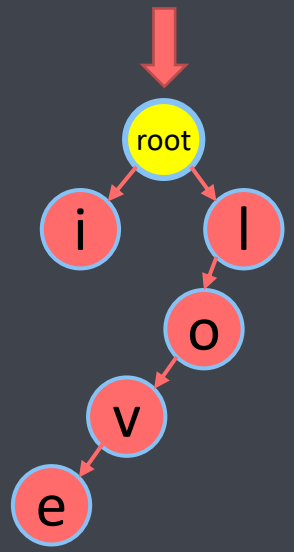
五. Code 最优解思路及编码实现

解法三：优化解法 桶排序+trie树

如输入[“i”, “love”, “leetcode”, “i”, “love”, “coding”] k = 2

统计每个单词出现频率后，将相同频率的单词放在同一个桶中。每个桶存一颗trie树





五. Code 最优解思路及编码实现

最优解：桶排序+trie树复杂度分析

时间复杂度：O(n)

- 遍历单词数据统计频率O(n)
- 每个单词构造trie树的时间为m，m为常数，总时间为O(mn)，O(mn) 等价于 O(n)
- O(n) +O(n)=O(2n)
- 忽略常数后：O(n)

空间复杂度：O(n)

- 存储每个单词出现频率O(n)
- 每个桶分配m的空间，总空间O(mn)，m为常数， O(mn) 等价于 O(n)
- O(n) +O(n)=O(2n)
- 忽略常数后：O(n)

五. Code 最优解思路及编码实现

解法三：桶排序+trie树参考代码

```
class Solution {
    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);
        }
        Trie[] buckets = new Trie[words.length];
        for(Map.Entry<String, Integer> e:map.entrySet()){
            //遍历每个单词，将其加入它所在桶的trie树
            String word = e.getKey();
            int freq = e.getValue();
            if(buckets[freq]==null){
                buckets[freq] = new Trie();
            }
            buckets[freq].addWord(word);
        }
        List<String> ans = new LinkedList<>();
        for(int i = buckets.length-1;i>=0;i--){
```

```
    //对于每个桶中的trie树，将单词按照字典序排序，和k比较
            if(buckets[i]!=null){
                List<String> l = new LinkedList<>();

                buckets[i].getWords(buckets[i].root, l);
                if(l.size()<k){
                    ans.addAll(l);
                    k = k - l.size();
                }
                else {
                   for(int j = 0;j<=k-1;j++){
                       ans.add(l.get(j));
                   }
                    break;
                }
            }
        }
        return ans;
    }
}
```

五. Code 最优解思路及编码实现

解法三：桶排序+trie树参考代码

Java编码实现

```
//构建trie树
class Trie {
    TrieNode root = new TrieNode();
    public void addWord(String word){
        TrieNode cur = root;
        for(char c:word.toCharArray()){
            if(cur.children[c-‘a’]==null){ cur.children[c-'a'] = new TrieNode();}
            cur = cur.children[c-'a'];
        }
        cur.word = word;
    }
   //对于每个桶中同样频率的单词，用DFS得到字典序顺序
    public void getWords(TrieNode root, List<String> ans){
        if(root ==null){return;}
        if(root.word!=null){ans.add(root.word);}
        for(int i = 0;i<=25;i++){
            if(root.children[i]!=null){ getWords(root.children[i], ans);}
        }
    }
     class TrieNode {//定义TrieNode
         TrieNode[] children;
         String  word;
         TrieNode() {
             this. children  = new TrieNode[26];
             this.word = null;
      }
}
```

执行结果： 通过 显示详情 >

执行用时： 9 ms ，在所有 Java 提交中击败了 46.20% 的用户

内存消耗： 39.6 MB ，在所有 Java 提交中击败了 5.09% 的用户

六. Change 变形延伸

题目变形

- （练习）最接近原点的 K 个点 (Leetcode 973)

延伸扩展

- 最大堆、最小堆在解决top k问题中有很强应用
- 当遇到排序问题时，考虑是否有其他办法避免全部排序，可以使问题简化

本题来源：

- leetcode 692 https://leetcode.com/problems/top-k-frequent-words/

总结

- 掌握字典（Map）的使用方法
- 掌握优先队列数据结构
- 掌握桶排序
- 掌握trie树的应用

课后练习

1. 数组中两个数的最大异或值 （Leetcode 421 /中等)
2. 最接近原点的K个点（ Leetcode 973 /中等 ）
3. 根据字符出现频率排序（ Leetcode 451 /中等 ）
4. 找出第K小的距离对（ Leetcode 719 /困难 ）

课后练习

1. 数组中两个数的最大异或值 (Leetcode 421 /中等)

提示：给定一个非空数组，数组中元素为 a0, a1, a2, … , an-1，其中 0 ≤ ai < 231 。

找到 ai 和aj 最大的异或 (XOR) 运算结果，其中0 ≤ i, j < n 。

你能在O(n)的时间解决这个问题吗？

```
输入: [3, 10, 5, 25, 2, 8]

输出: 28

解释: 最大的结果是 5 ^ 25 = 28.
```

课后练习

2. 最接近原点的K个点（Leetcode 973/中等）

提示：1 <= K <= points.length <= 10000 ； -10000 < points[i][0] < 10000

-10000 < points[i][1] < 10000

输入: points = [[1,3],[-2,2]], K = 1
输出: [[-2,2]]
解释:
(1, 3) 和原点之间的距离为 sqrt(10),
(-2, 2) 和原点之间的距离为 sqrt(8),
由于 sqrt(8) < sqrt(10), (-2, 2) 离原点更近。
我们只需要距离原点最近的 K = 1 个点，所以答案就是
[[-2,2]]。

输入: points = [[3,3],[5,-1],[-2,4]], K = 2
输出: [[3,3],[-2,4]]
（答案 [[-2,4],[3,3]] 也会被接受。）

课后练习

3. 根据字符出现频率排序（Leetcode 451 /中等）

提示：给定一个字符串，请将字符串里的字符按照出现的频率降序排列。

输入：
"tree"

输出：
"eert"

解释：
'e'出现两次，'r'和't'都只出现一次。
因此'e'必须出现在'r'和't'之前。此外，"eetr"也是一个有效的答案。

输入：
"cccaaa"

输出：
"cccaaa"

解释：
'c'和'a'都出现三次。此外，"aaaccc"也是有效的答案。
注意"cacaca"是不正确的，因为相同的字母必须放在一起。

课后练习

4. 找出第K小的距离对（Leetcode 719/困难）

提示：2 <= len(nums) <= 10000； 0 <= nums[i] < 1000000.

1 <= k <= len(nums) * (len(nums) - 1) / 2.

输入：
nums = [1,3,1]
k = 1
输出：0
解释：
所有数对如下：
(1,3) -> 2
(1,1) -> 0
(3,1) -> 2
因此第 1 个最小距离的数对是 (1,1)，它们之间的距离为 0。

拉勾教育

—互联网人实战大学—

下载「拉勾教育App」
获取更多内容