入选理由

- 复习哈希表

标签

- 链表
- 堆

难度

- 中等

题目地址（451 根据字符出现频率排序）

https://leetcode-cn.com/problems/sort-characters-by-frequency/comments/

题目描述

```
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。

示例 1:

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。
示例 2:

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。
示例 3:

输入:
"Aabb"

输出:
"bbAa"

解释:
此外，"bbaA"也是一个有效的答案，但"Aabb"是不正确的。
注意'A'和'a'被认为是两种不同的字符。
```

前置知识

- 排序算法
- 堆
- 哈希表

方法一 直接排序

思路

基本记录就是统计字符个数，再把个数排个序就可以，采用哈希表的方式存储每个字符的出现次数，然后选择个排序算法。

- 用哈希表统计每个字符的出现次数
- 按每个字符出现的次数调用库函数进行排序
- 将排序后的字符进行拼接

代码

代码支持：Python3， Java，CPP

Python3 Code：

```python
class Solution:
    def frequencySort(self, s: str) -> str:

        dict = {}
        for ch in s:
            dict[ch] = dict.get(ch, 0) + 1
```

```
        vals = sorted(dict.items(), key=lambda x : x[1], reverse=True)

        res = ""

        for k, v in vals:
            res += k * v

        return res
```

Java Code：

```
class Solution {

    public String frequencySort(String s) {

        Map<Character, Integer> counter = new HashMap<>();

        for (int i = 0; i < s.length(); i++)
            counter.put(s.charAt(i), counter.getOrDefault(s.charAt(i), 0) + 1);

        List<Map.Entry<Character, Integer>> list = new ArrayList<>(counter.entrySet());

        Collections.sort(list, new Comparator<Map.Entry<Character, Integer>>() {

            @Override
            public int compare(Map.Entry<Character, Integer> o1, Map.Entry<Character, Integer> o2) {

                return o2.getValue() - o1.getValue();
            }
        });

        StringBuilder res = new StringBuilder();

        for (Map.Entry<Character, Integer> entry : list)
            for (int i = 0; i < entry.getValue(); i++)
                res.append(entry.getKey());

        return res.toString();
    }
}
```

CPP Code：

```
class Solution {
public:
    string frequencySort(string s) {
        unordered_map<char, int> mp;
```

```
        int length = s.length();
        for (auto &ch : s) {
            mp[ch]++;
        }
        vector<pair<char, int>> vec;
        for (auto &it : mp) {
            vec.emplace_back(it);
        }
        sort(vec.begin(), vec.end(), [](const pair<char, int> &a, const pair<char, int> &b) {
            return a.second > b.second;
        });
        string ret;
        for (auto &[ch, num] : vec) {
            for (int i = 0; i < num; i++) {
                ret.push_back(ch);
            }
        }
        return ret;
    }
};
```

复杂度分析

设N为字符个数，K为去重字符个数

- 时间复杂度：$O(N + KlogK)$
- 空间复杂度：直接排序：$O(K)$

进阶: 是否可以实现$O(n + k)$时间复杂度呢（提示：桶排序）

方法二 - 堆排序

思路

思路与法一一致，不过这里不使用库里自带的排序。而是自己手写实现。

具体算法：

1. 新建一个大顶堆
2. 循环字符串，将所有元素入堆
3. 利用大顶堆的性质，对堆进行取顶（得到的是字符串中出现频率最高的字符）
4. 拼接字符串，重复步骤 3 - 4 直到堆为空

代码

代码支持：Java

Java Code：

```java
class Solution {

    public String frequencySort(String s) {

        Map<Character, Integer> map = new HashMap();
        PriorityQueue<Character> pq = new PriorityQueue<>((a, b) -> map.get(b) - map.get(a));

        for (int i = 0; i < s.length(); i++)
            map.put(s.charAt(i), map.getOrDefault(s.charAt(i), 0) + 1);

        for (char ch : map.keySet())
            pq.offer(ch);

        StringBuilder res = new StringBuilder();

        while(!pq.isEmpty()){

            char c = pq.poll();
            int count = map.get(c);

            for (int i = 0; i < count; i++)
                res.append(c);

        }

        return res.toString();
    }
}
```

复杂度分析

设N为字符个数，K为去重字符个数

- 时间复杂度：$O(N + KlogK)$

- 空间复杂度：$O(K)$

上一页 下一页