## 总结

1. 就像有人评论说的，这次比赛比的是做题速度。本次比赛前三题的难道并不大，尤其是第一个中等题的暴力解法简直白给，第二个中等题也达不到应有的难度，因此可以看出，比赛之中首重速度的重要性，只要能AC，不管复杂度如何，代码码的快最重要
2. 困难题确实很有难度，做了一个多小时，把自己都绕晕了，只能说还是缺乏高等级的编程练习。

## 1394. 找出数组中的幸运数（简单）

1. 题目描述

   在整数数组中，如果一个整数的出现频次和它的数值大小相等，我们就称这个整数为「幸运数」。

   给你一个整数数组 arr，请你从中找出并返回一个幸运数。

   如果数组中存在多个幸运数，只需返回 最大 的那个。 如果数组中不含幸运数，则返回 -1 。

   ```
   示例 1:
   输入: arr = [2,2,3,4]
   输出: 2
   解释: 数组中唯一的幸运数是 2 ，因为数值 2 的出现频次也是 2 。

   示例 2:
   输入: arr = [1,2,2,3,3,3]
   输出: 3
   解释: 1、2 以及 3 都是幸运数，只需要返回其中最大的 3 。

   示例 3:
   输入: arr = [2,2,2,3,3]
   输出: -1
   解释: 数组中不存在幸运数。

   示例 4:
   输入: arr = [5]
   输出: -1

   示例 5:
   输入: arr = [7,7,7,7,7,7,7]
   输出: 7
   ```

提示:

- 1 <= arr.length <= 500
- 1 <= arr[i] <= 500

2. 比赛实现

   知道map可能程序会更高效，但是比赛要赶时间，所以直接for循环代码最简单，map的方法就不再写了

```
class Solution {
public:
    int findLucky(vector<int>& arr) {
        vector<int> cnt(501, 0);
        for(int i = 0; i < arr.size(); i++)
            cnt[arr[i]]++;
        for(int i = 500; i > 0; i--)
            if(cnt[i] == i)
                return i;
        return -1;
    }
};
```

## 1395. 统计作战单位（中等）

1. 题目描述

   n 名士兵站成一排。每个士兵都有一个 独一无二 的评分 rating 。每 3 个士兵可以组成一个作战单位，分组规则如下：

   - 从队伍中选出下标分别为 i、j、k 的 3 名士兵，他们的评分分别为 rating[i]、rating[j]、rating[k]
   - 作战单位需满足： rating[i] < rating[j] < rating[k] 或者 rating[i] > rating[j] > rating[k] ，其中 0 <= i < j < k < n

   请你返回按上述条件可以组建的作战单位数量。每个士兵都可以是多个作战单位的一部分。

```
示例 1：
输入：rating = [2,5,3,4,1]
输出：3
解释：我们可以组建三个作战单位 (2,3,4)、(5,4,1)、(5,3,1) 。

示例 2：
输入：rating = [2,1,3]
输出：0
解释：根据题目条件，我们无法组建作战单位。

示例 3：
输入：rating = [1,2,3,4]
输出：4
```

提示：

- n == rating.length
- 1 <= n <= 200
- 1 <= rating[i] <= 10^5

2. 比赛实现

   比赛时根据题目数据中n的范围，暴力循环O(n^3)的复杂度就可以满足要求了，但赛后思考发现，可以用O(n^2)解决，思路如下：

   - 假设以某个士兵j为中间的士兵，分别统计其 左/右 侧评分 大于/小于 rating[j]的士兵数，则以士兵j为中间的士兵可以组成的作战单位数量为smaller_cnt_l*bigger_cnt_r* + *bigger_cnt_l*smaller_cnt_r;

- 遍历所有的士兵，以该士兵为中间，统计并加和其可以组成的作战单位数量，整体遍历一遍后，所有的组合结果肯定无重复，因此这就是答案

```
class Solution {
public:
    int numTeams(vector<int>& rating) {
        int n = rating.size();
        if(n < 3) return 0;
        int ans = 0;
        for(int j = 1; j < n-1; j++){//以j为中间
            int smaller_cnt_l = 0;
            int bigger_cnt_l = 0;
            int smaller_cnt_r = 0;
            int bigger_cnt_r = 0;
            for(int i = 0; i < j; i++){//左侧
                if(rating[i] < rating[j])
                    smaller_cnt_l++;
                else
                    bigger_cnt_l++;
            }
            for(int k = j+1; k < n; k++){//右侧
                if(rating[k] < rating[j])
                    smaller_cnt_r++;
                else
                    bigger_cnt_r++;
            }
            ans += smaller_cnt_l*bigger_cnt_r + bigger_cnt_l*smaller_cnt_r;
        }
        return ans;
    }
};
```

3. 离散化树状数组——O(nlogn)

   先了解一下树状数组：

   - https://zhuanlan.zhihu.com/p/25185969
   - https://www.zhihu.com/question/54404092

### 方法三：离散化树状数组

#### 前置知识

- 离散化思想，在不改变数据相对大小的条件下，对数据进行相应的缩小。
- 树状数组（二元索引树），一种动态维护前缀和的数据结构。

#### 思路

考虑优化方法二中求 $i_{less}$、$k_{more}$、$i_{more}$、$k_{less}$ 的过程。在方法二中我们使用了枚举来求解这四个量，单次枚举的时间代价是 $O(N)$。假设我们有一个桶数组，索引 $i$ 的值为 $1$ 就说明存在元素 $i$，为 $0$ 就说明不存在元素 $i$，那么该桶数组的前缀和 $\mathrm{preffixSum}[i-1]$ 就表示当前比 $i$ 小的数的个数，我们只需要用树状数组动态维护这个前缀和，就可以把单次的时间代价从 $O(N)$ 优化到 $O(\log N)$。

我们对 `rating` 数组做两次遍历，一次从前向后，一次从后向前。从前向后的时候，对于每一个 `rating[i]`（记为 $x$），求到上述桶数组下标 $x-1$ 的前缀和，即 $i_{less}$，记 `rating` 数组中出现的最大值为 $r_{\max}$，用 $r_{\max}$ 的前缀和减去 $x$ 位置的前缀和即可得到 $i_{more}$。从后向前的那次遍历同理。

**思考：仅仅这样做真的可以单次计算变成 $\log N$ 吗？** 我们知道树状数组修改和查询的时间代价和树状数组的长度相关，也就是这里的 $r_{\max}$（它最大可以到 $10^5$），所以这里单次查询的代价是 $O(\log r_{\max})$。实际上 `rating` 的长度最大只有 $200$，也就是这个树状数组中的「有效位置」最多只有 $200$ 个，所以我们不用开辟 $10^5$ 的长度，只需要开辟 $200$ 的长度，通过离散化的方法缩小值域，这样就可以把单次的时间代价变成 $O(\log N)$。

由于这里没有重复的数字，所以只需要对 `rating` 数组中的数进行排序，然后二分获取离散化之后的值即可，单次二分的时间代价也是 $O(\log N)$。

```
class Solution {
public:
    static constexpr int MAX_N = 200 + 5;

    int c[MAX_N];
    vector <int> disc;
    vector <int> iLess, iMore, kLess, kMore;

    int lowbit(int x) {
        return x & (-x);
    }

    void add(int p, int v) {
        while (p < MAX_N) {
            c[p] += v;
            p += lowbit(p);
        }
    }

    int get(int p) {
        int r = 0;
        while (p > 0) {
            r += c[p];
            p -= lowbit(p);
        }
        return r;
    }
```

```
    int numTeams(vector<int>& rating) {
        disc = rating;
        disc.push_back(-1);
        sort(disc.begin(), disc.end());
        auto getId = [&] (int target) {
            return lower_bound(disc.begin(), disc.end(), target) - disc.begin();
        };


        iLess.resize(rating.size());
        iMore.resize(rating.size());
        kLess.resize(rating.size());
        kMore.resize(rating.size());

        for (int i = 0; i < rating.size(); ++i) {
            auto id = getId(rating[i]);
            iLess[i] = get(id);
            iMore[i] = get(201) - get(id);
            add(id, 1);
        }

        memset(c, 0, sizeof c);
        for (int i = rating.size() - 1; i >= 0; --i) {
            auto id = getId(rating[i]);
            kLess[i] = get(id);
            kMore[i] = get(201) - get(id);
            add(id, 1);
        }

        int ans = 0;
        for (unsigned i = 0; i < rating.size(); ++i) {
            ans += iLess[i] * kMore[i] + iMore[i] * kLess[i];
        }

        return ans;
    }
}
```

## 1396. 设计地铁系统（中等）

1. 题目描述

   请你实现一个类 UndergroundSystem ， 它支持以下 3 种方法：

   1. checkIn(int id, string stationName, int t)
      - 编号为 id 的乘客在 t 时刻进入地铁站 stationName 。
      - 一个乘客在同一时间只能在一个地铁站进入或者离开。
   2. checkOut(int id, string stationName, int t)
      - 编号为 id 的乘客在 t 时刻离开地铁站 stationName 。
   3. getAverageTime(string startStation, string endStation)

- 返回从地铁站 startStation 到地铁站 endStation 的平均花费时间。
- 平均时间计算的行程包括当前为止所有从 startStation 直接到达 endStation 的行程。
- 调用 getAverageTime 时，询问的路线至少包含一趟行程。

你可以假设所有对 checkIn 和 checkOut 的调用都是符合逻辑的。也就是说，如果一个顾客在 t1 时刻到达某个地铁站，那么他离开的时间 t2 一定满足 t2 > t1 。所有的事件都按时间顺序给出。

```
示例：
输入：
"UndergroundSystem","checkIn","checkIn","checkIn","checkOut","checkOut","checkOut",
"getAverageTime","getAverageTime","checkIn","getAverageTime","checkOut","getAverage
Time",[45,"Leyton",3],[32,"Paradise",8],[27,"Leyton",10],[45,"Waterloo",15],
[27,"Waterloo",20],[32,"Cambridge",22],["Paradise","Cambridge"],
["Leyton","Waterloo"],[10,"Leyton",24],["Leyton","Waterloo"],[10,"Waterloo",38],
["Leyton","Waterloo"]]
输出：
[null,null,null,null,null,null,null,14.0,11.0,null,11.0,null,12.0]
解释：
UndergroundSystem undergroundSystem = new UndergroundSystem();
undergroundSystem.checkIn(45, "Leyton", 3);
undergroundSystem.checkIn(32, "Paradise", 8);
undergroundSystem.checkIn(27, "Leyton", 10);
undergroundSystem.checkOut(45, "Waterloo", 15);
undergroundSystem.checkOut(27, "Waterloo", 20);
undergroundSystem.checkOut(32, "Cambridge", 22);
undergroundSystem.getAverageTime("Paradise", "Cambridge");       // 返回 14.0。从
"Paradise" (时刻 8) 到 "Cambridge"(时刻 22)的行程只有一趟
undergroundSystem.getAverageTime("Leyton", "Waterloo");          // 返回 11.0。总共有
2 躺从 "Leyton" 到 "Waterloo" 的行程，编号为 id=45 的乘客出发于 time=3 到达于 time=15，编
号为 id=27 的乘客于 time=10 出发于 time=20 到达。所以平均时间为 ( (15-3) + (20-10) ) / 2
= 11.0
undergroundSystem.checkIn(10, "Leyton", 24);
undergroundSystem.getAverageTime("Leyton", "Waterloo");          // 返回 11.0
undergroundSystem.checkOut(10, "Waterloo", 38);
undergroundSystem.getAverageTime("Leyton", "Waterloo");          // 返回 12.0
```

提示：

- 总共最多有 20000 次操作。
- 1 <= id, t <= 10^6
- 所有的字符串包含大写字母，小写字母和数字。
- 1 <= stationName.length <= 10
- 与标准答案误差在 10^-5 以内的结果都视为正确结果。

2. 比赛实现

```
class UndergroundSystem {
public:
    unordered_map<int, pair<string, int>> passenger;//<乘客id, <上车站, 上车时间>>
    unordered_map<string, pair<int, int>> time;//<"上车站,下车站", <总用时, 总人次>>
    UndergroundSystem() {
    }
    void checkIn(int id, string stationName, int t) {
```

```
            passenger[id] = make_pair(stationName, t);
        }
        void checkOut(int id, string stationName, int t) {
            string s2e = passenger[id].first + ',' + stationName;
            int time_spend = t - passenger[id].second;
            if(time.find(s2e) == time.end())
                time[s2e] = make_pair(time_spend, 1);
            else{
                time[s2e].first += time_spend;
                time[s2e].second += 1;
            }
        }
        double getAverageTime(string startStation, string endStation) {
            string s2e = startStation + ',' + endStation;
            double ans =  double(time[s2e].first) / time[s2e].second;
            return ans;
        }
    };
```

## 1397. 找到所有好字符串（困难）

1. 题目描述

给你两个长度为 `n` 的字符串 `s1` 和 `s2` ，以及一个字符串 `evil` 。请你返回 **好字符串** 的数目。

**好字符串** 的定义为：它的长度为 `n` ，字典序大于等于 `s1` ，字典序小于等于 `s2` ，且不包含 `evil` 为子字符串。

由于答案可能很大，请你返回答案对 10^9 + 7 取余的结果。

**示例 1：**

```
输入：n = 2, s1 = "aa", s2 = "da", evil = "b"
输出：51
解释：总共有 25 个以 'a' 开头的好字符串："aa", "ac", "ad", ..., "az"。还有 25 个以 'c' 开头的好字符串："ca", "cc", "cd", ..., "cz"。最后，还有一个以 'd' 开头的好字符串："da"。
```

**示例 2：**

```
输入：n = 8, s1 = "leetcode", s2 = "leetgoes", evil = "leet"
输出：0
解释：所有字典序大于等于 s1 且小于等于 s2 的字符串都以 evil 字符串 "leet" 开头。所以没有好字符串。
```

**示例 3：**

```
输入：n = 2, s1 = "gx", s2 = "gz", evil = "x"
输出：2
```

**提示：**

- `s1.length == n`
- `s2.length == n`
- `1 <= n <= 500`
- `1 <= evil.length <= 50`
- 所有字符串都只包含小写英文字母。

2. 正确解法

都说是KMP+DP，我他妈，还是，看不懂、、、似懂非懂、、、，贴个最看得懂的吧、、

观察题目，发现题中所求字符串有三个要求：

1. s字典序大于等于s1
2. s字典序小于等于s2
3. s不包含evil

设计DP状态来表示这三个要求，F[i][j][k=0/1][l=0/1]表示：
已经构造了s的前i个字符，
且匹配到evil的前j个字符；
k=0表示字典序已经比s1大，k=1表示字典序等于s1的前i个字符组成的前缀
l同理。

考虑主动转移，即现在状态是F[i][j][k][l],枚举下一个字符a，看看能否转移到F[i+1][p][x][y]：
这三个状态转移是分开的。
j->p的转移相当于字符串匹配，用kmp处理，当p=m时，这个状态就非法了，就不管了。
考虑k的转移，l同理：

1. k=0,则x=0
2. k=1
   若a==s1[i],x=1
   若a>s1[i],x=0
   若a<s1[i],非法

ps:代码中所用符号与题解不同

```
class Solution {
public:
    int findGoodStrings(int n, string s1, string s2, string evil) {
        if(s1 > s2) return 0;
        int MOD=1e9+7;
        long long F[510][51][2][2];
        int m=evil.size();
        memset(F,0,sizeof F);
        int Next[100];
        Next[0]=-1;
        for (int i=0,j=-1;i<m;) {
            if (j==-1 || evil[i]==evil[j]) ++i,++j,Next[i]=j;
```

```
            else j=Next[j];
        }
        int a,b;
        a=s1[0]-'a';
        b=s2[0]-'a';
        for (int i=0;i<26;++i) {
            if (i<a || i>b) continue;
            bool f1=false,f2=false;
            if (i==a) f1=true;
            if (i==b) f2=true;
            int p=0;
            if (i==evil[0]-'a') p=1;
            if (p==m) continue;
            F[1][p][f1][f2]++;
        }
        for (int i=1;i<n;++i) {
            a=s1[i]-'a';
            b=s2[i]-'a';
            for (int j=0;j<26;++j)//取j
                for (int k=0;k<m;++k) {//匹配evil的前k个字符
                    int p=k;
                    while (p!=-1) {
                        if (evil[p]-'a'==j) {++p;break;}
                        p=Next[p];
                    }
                    if (p==-1) p=0;
                    if (p==m) continue;
                    for (int f1=0;f1<2;++f1)
                        for (int f2=0;f2<2;++f2) {
                            int x,y;
                            if (f1==0) x=0;
                            else {
                                if (j==a) x=1;
                                else if (j>a) x=0;
                                else continue;
                            }
                            if (f2==0) y=0;
                            else {
                                if (j==b) y=1;
                                else if (j<b) y=0;
                                else continue;
                            }
                            F[i+1][p][x][y]+=F[i][k][f1][f2];
                        }
                }
            for (int k=0;k<m;++k)
                for (int x=0;x<2;++x)
                    for (int y=0;y<2;++y)
                        F[i+1][k][x][y]%=MOD;
        }
        long long Ans=0;
        for (int i=0;i<m;++i) {
            Ans+=F[n][i][0][0];
```

```
            Ans+=F[n][i][0][1];
            Ans+=F[n][i][1][0];
            Ans+=F[n][i][1][1];
        }
        Ans%=MOD;
        return (int)Ans;
    }
};
```