LeetCode 1394. 找出数组中的幸运数

1394. 找出数组中的幸运数

难度 简单 2

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
```

```cpp
class Solution {
public:
    int findLucky(vector<int>& arr) {
        int res = -1;
        map<int,int> cnt;
        for(int i:arr) cnt[i] ++;
        for(auto p:cnt) if(p.first == p.second) res = p.first;

        return res;
    }
};
```

LeetCode 1395. 统计作战单位数

1395. 统计作战单位数

难度 中等 8

`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`

```
class Solution {
public:
    int numTeams(vector<int>& rating) {
        int n = rating.size();
        int ans = 0;
        for(int i = 0;i < n;i ++)
        {
            for(int j = i + 1;j < n;j ++)
            {
                if(rating[i] >= rating[j]) continue;
                for(int k = j + 1;k < n;k ++)
                    if(rating[j] < rating[k]) ans ++;
            }
            for(int j = i + 1;j < n;j ++)
            {
```

```
                if(rating[i] <= rating[j]) continue;
                for(int k = j + 1;k < n;k ++)
                    if(rating[j] > rating[k]) ans ++;
            }
        }
        return ans;
    }
};
```

LeetCode 1396. 设计地铁系统

1396. 设计地铁系统

难度 中等 5

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 t_1 时刻到达某个地铁站，那么他离开的时间 t_2 一定满足 $t_2 > t_1$ 。所有的事件都按时间顺序给出。

示例：

输入：

```
["UndergroundSystem","checkIn","checkIn","checkIn","checkO
[[],[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,nu
```

```
class UndergroundSystem {
public:
```

```
    typedef pair<string,string> PSS;
    map<PSS,double> ma;
    map<PSS,int> cnt;
    unordered_map<int,string> pre;
    unordered_map<int,double> pt;
    UndergroundSystem() {

    }

    void checkIn(int id, string stationName, int t) {
        pre[id] = stationName;
        pt[id] = t;
    }

    void checkOut(int id, string stationName, int t) {
        if(pre.count(id))
        {
            PSS p(pre[id],stationName);
            ma[p] += t - pt[id];
            cnt[p] ++;
        }
    }

    double getAverageTime(string startStation, string endStation) {
        PSS p(startStation,endStation);

        return ma[p]/cnt[p];
    }
};

/**
 * Your UndergroundSystem object will be instantiated and called as such:
 * UndergroundSystem* obj = new UndergroundSystem();
 * obj->checkIn(id,stationName,t);
 * obj->checkOut(id,stationName,t);
 * double param_3 = obj->getAverageTime(startStation,endStation);
 */
```

LeetCode 1397. 找到所有好字符串

1392. 最长快乐前缀

难度 困难 13

「快乐前缀」是在原字符串中既是 **非空** 前缀也是后缀（不包括原字符串自身）的字符串。

给你一个字符串 `s`，请你返回它的 **最长快乐前缀**。

如果不存在满足题意的前缀，则返回一个空字符串。

示例 1：

```
输入：s = "level"
输出："l"
解释：不包括 s 自己，一共有 4 个前缀（"l", "le", "lev", "leve"）和 4 个后缀（"l", "el", "vel", "evel"）。最长的既是前缀也是后缀的字符串是 "l" 。
```

示例 2：

```
输入：s = "ababab"
输出："abab"
解释："abab" 是最长的既是前缀也是后缀的字符串。题目允许前后缀在原字符串中重叠。
```

示例 3：

```
输入：s = "leetcodeleet"
输出："leet"
```

示例 4：

```
输入：s = "a"
输出：""
```