信息学奥赛笔记12

点考试题讲解，01-20期末考试T4

[U398698]时间安排 https://www.luogu.com.cn/problem/U398698

点考试题T1

题目描述

现在有n个比赛的候选人，我们现在要为他们举办比赛。每个候选人都有一个连续的空闲时间来参加比赛，对于每一个候选人，都有一个$[l_i, r_i]$的空闲时间。现在的问题是，有没有一个时间，所有人都有空。这样我们就可以在那个时候安排比赛。

输入格式

第一行包含一个整数$T(1 \leq T \leq 10^3)$，表示有T组测试用例。测试用例的说明如下：

对于每个测试用例，第一行包含一个整数$n(1 \leq n \leq 10^3)$个候选人。

下一个n行，每行两个整数。l_i、$r_i(1 \leq l_i, r_i \leq 10^3)$，表示$i$候选人的空闲时间段。

可以保证所有的$n \leq 10^3$，适用于所有用例。

输出格式

如果所有人都有一个空闲的时间，则在一行中输出"YES"，否则就输出"NO"；

样例 #1

样例输入 #1

```
2
3
1 2
1 3
2 3
2
5 10
1 4
```

样例输出 #1

```
YES
NO
```

提示

对于示例中第一个样例，每个人都有空的时间是2。

对于示例中第二个样例，没有每个人都有空的时间。

思路

有关这道题的第一个难点，是大家没有见过类似这种T组样例的输入输出，这种输入输出方式，在大型竞赛中是非常常见的，各位同学们可以设想一下，如果一个测试样例的答案就是YES或者NO，那只需要直接输出YES或者No就可以骗到不少分，这肯定是出题的人不希望看到的结果，那该怎么解决呢，我们可以把很多个样例绑到一起，作为一个测试点，也就是说，我们需要运行代码T次，作为一个测试点给一个测试点的分，这样就可以避免直接输出YES或者直接输出NO就能拿到分的情况。

有关这种T组样例的题目，老师给大家一个模板

```
#include <bits/stdc++.h>
using namespace std;
int main() {
    int t;
    cin >> t;
    while (t--) {
        //变量定义在这，代码也在这写。
    }
    return 0;
}
```

各位同学要注意一个细节，为什么变量需要定义在函数内，而不是全局变量区了？

因为第一次在跑完整个程序后，第二次，你的那些全局变量没有重置成他应该重置的值，这种类型的题目各位同学一定要记得转变一下思维。要有把变量变成临时变量使用的能力，这里老师给大家再写一个模板。

```
#include <bits/stdc++.h>
using namespace std;

void solve() {
    int n;
    cin >> n;
    int nums[n];
    for (int i = 0; i < n; i++) {
        cin >> nums[i];
    }
}

int main() {
    int t;
    cin >> t;
    while (t--) {
        solve();
    }
    return 0;
}
```

这个模板是一个更通用的T组样例的模板，我们可以写一个solve函数，solve的意思是解决，解决问题，既然程序让我们写出T组样例的答案，一行一个，那我们就把这T组看成是程序运行了T遍，每一次输出完后换行，原本应该写到main函数的代码，我们放到solve函数来写，全局变量全部放到solve函数的最顶部来定义，这样的话，我们写代码的时候只需要关注solve函数即可，不需要关心其他的部分。关于这两个模板，希望大家能够选取一个适合自己的，并把他背掉，在后面我们给大家出的题目当中，可能会有不少题目都需要使用T组样例输入输出的形式。

好，我们回到这道题来，看这组样例的第一个样例。

我们用一个表来表示这三个人的空闲时间

编号i	1	2	3
0	√	√	
1	√	√	√
2		√	√

题目让我们找寻的就是，存不存在某一列，全打上了√。

大家可以关注一下这两组区间[1, 2][1,3]他们是不是具有一个公共的区间[1,2]，那么假如这n个人他们存在一个公共的区间的话，我们让他们两两相交，应该会有一个公共的部分吧。比如说区间[1,5]和区间[3, 8]，他们就有一个公共的部分[3, 5]，那我们是不是可以求一下这n个区间最终公共的部分是多少，不就知道他们n个人是不是具有共同的空闲时间了吗。那这个公共部分的[3,5]是怎么求出来的呢，我们是不是可以取两个区间的左区间的较大值max(1,3) 作为最终左区间[3, _]取两个区间右区间的较小值min(5,8)作为最终右区间[_, 5],合起来就是[3, 5]。

我们再举一个例子：区间[1, 2]和区间[3,4],那按照刚刚的想法我们取他们的共同区间为[max(1,3), min(2,4)] = [3, 2]这是一个合理的区间吗?

很明显它不是的，那也就是说，只要最后我们发现，我们获得的公共区间是个左区间 > 右区间，是不是就说明，无解，应该输出No，那如果最终咱们发现是一个合理的区间，也就是左区间小于右区间，就说明有解，输出YES。代码如下：

```cpp
#include <bits/stdc++.h>
using namespace std;
int main() {
    int t;
    cin >> t;
    while (t--) {
        int n, x = 0, y = INT_MAX, l, r; //变量定义在函数内，这样每次到新的一轮输入数据的时候，这些变量的值就被重置了
        cin >> n;//输入n个候选人
        while (n--) {
            cin >> l >> r;//这是当前这个人的空闲时间区间
            x = max(x, l);//和答案的左区间取较大值
            y = min(y, r);//和答案的右区间取较小值
        }
        if (x > y) cout << "NO" << endl; //如果答案的左区间大于右区间，不合法，输出NO
        else cout << "YES" << endl;  //如果最终答案是一个合法的时间区间，开始小于等于结束，输出YES
    }
    return 0;
```

```
}
```

[U399301]卡牌游戏 https://www.luogu.com.cn/problem/U399301

点考试题T2

题目描述

小A和小B在玩一款新型的双人对战桌游。具体规则如下：

首先小A先抽取n张卡牌，第i张卡牌上有一个胜利点数a_i。然后小B再抽取m张卡牌，第i张卡牌上有一个胜利点数b_i。这个游戏一共有k轮，对于每个$i = 1, 2, 3 \ldots k$，他们都按照以下的具体规则操作：

- 如果i是奇数，那么小A可以选择一张卡牌与小B进行交换，或者什么都不做
- 如果i是偶数，那么小B可以选择一张卡牌与小A进行交换，或者什么都不做

小A和小B都是这个世界上最聪明的人，他们在每一次操作的时候都会采用最优策略。现在需要你计算出k轮之后，**小A**的最大胜利点数。

输入格式

第一行包含三个整数n, m, k，分别表示小A、小B的卡牌个数以及对战回合数。

接下来的一行，有n个整数，分别代表小A的初始卡牌胜利点数。

接下来的一行，有m个整数，分别代表小B的初始卡牌胜利点数。

输出格式

对于每个测试用例，输出一个整数，表示小A在游戏结束时可以获得的最大胜利点数。

样例 #1

样例输入 #1

```
2 2 1
1 2
3 4
```

样例输出 #1

```
6
```

样例 #2

样例输入 #2

```
1 1 10000
1
2
```

样例输出 #2

```
1
```

提示

在第一个测试样例中，小A将交换点数为1和4的卡牌。

在第二个测试样例中，两位玩家将交换这两个卡牌1万次。

$1 \leq T \leq 100$，$1 \leq n \leq m \leq 10^3$，$1 \leq k \leq 10^5$，$1 \leq a_i, b_i \leq 10^9$。

思路

在这道题中，我们把题目简化一下，有俩数组，一共进行k轮，在奇数轮的时候轮到A操作，在偶数轮的时候轮到B操作，那么他们每一次就会选择把自己最小的那个牌去换对方最大的那个牌，也可以不换，问经过K轮后，A这个玩家最多能获得多少分。

那既然涉及到了最小和最大，不妨给AB都排个序，我们就按照题目的意思，把答案模拟出来。

那么各位同学可以想一下，什么情况下，才需要拿我最小的去跟他最大的换，是不是只有在他最大的那个比我最小的那个大的情况对吧，那如果对方最大的那张牌都小于我最小的那一张，我换是不是就亏了，就选择不换呗。

90分做法

```cpp
#include <bits/stdc++.h>
using namespace std;
int a[1005], b[1005], n, m, k;
long long sum;
int main() {
    cin >> n >> m >> k;
    for (int i = 0; i < n; i++) cin >> a[i];
    for (int j = 0; j < m; j++) cin >> b[j];
    sort(a, a + n);
    sort(b, b + m);//给a， b排序
    for (int i = 1; i <= k; i++) {//游戏一共进行k轮
        if (i % 2 == 1) { //如果第i轮是个奇数轮，则轮到a操作
            if (a[0] < b[m - 1]) { //如果a最小的那张牌小于b最大的那张牌，就应该换
                swap(a[0], b[m - 1]); //换牌
            }
        }
        if (i % 2  == 0) { //如果第i轮是个偶数轮，则轮到b操作
            if (b[0] < a[n - 1]) { //如果b最小的那张牌小于a最大的那张牌，就应该换
                swap(b[0], a[n - 1]);//换牌
            }
        }
        sort(a, a + n);//每次换完之后还需要对a，b再排序一次
        sort(b, b + m);
```

```
    }
    for (int i = 0 ; i < n; i++) {
        sum += a[i];//最后咱们把a的得分加起来就是答案，记得开long long
    }
    cout << sum << endl;
    return 0;
}
```

> 时间复杂度$O(knlogn)$，k是游戏进行的轮数，每一轮需要对a,b排序一次，时间复杂度为$O(nlogn)$。
>
> 空间复杂度为$O(n+m)$，n是A的长度，m是B的长度，我们需要存储他们的分。

令人震惊的是，就这样按照题目的意思把代码模拟出来，居然可以拿到**90**分！！！

只需要按照题目的意思模拟出答案就有那么高的分数！为什么你们都拿不到！！！

100分做法

好那么大家用模拟的方式做完题后你们可以思考一下，一开始，

A拿到的牌有哪些？经过排序后是

[A_{min}，**中间一坨**，A_{max}]

B拿到的牌呢？

[B_{min}，**中间一坨**，B_{max}]

那第一轮的时候A要做什么操作呢？他是不是要拿A_{min}去和B_{max}比较一下，此时有两种情况

第**1**种情况：

如果$A_{min} \geq B_{max}$，说明什么，A拿到的都是大牌，他换不换牌，他肯定不换啊。

那对于第**2**轮开始，到B玩家操作了，他怎么办啊？是不是肯定拿B_{min}去换A_{max}。

那再接下来呢？A想不想换回来，想啊！所以他们俩是不是就在对这两张牌换来换去，谁最后一次操作，谁就能获得大牌，所以当k是个奇数时，最后都是A玩家拿到大牌，也就相当于只进行了1轮，没有任何的换牌，如果k是个偶数，说明最后操作的人是玩家B，也就是相当于游戏只进行了2轮。

第**2**种情况：

如果$A_{min} < B_{max}$，此时，在第1轮中，你是A你换不换牌？肯定换，换了收益更高。

那么现在A玩家是不是拿到了两张大牌[A_{max}，B_{max}]

B玩家拿到了两张小牌[A_{min}，B_{min}]

轮到B玩家操作的时候，他一定会怎么办？拿这两张小牌中最小的那个，去换两张大牌中最大的那个，那么再往后呢，A就希望换回来了对吧？

所以从第2轮开始，这两个人就已经开始对他们两个当中最小的那张牌，和最大的那张牌来回交换了！

那么如果k是个奇数，说明最后一次是A操作，也就相当于进行了1轮，他可以把最小的那张牌踢给B，然后自己拿着2张大牌；

如果k是个偶数，说明最后一次是B操作，也就相当于进行了2轮，他可以把最小的那张牌踢给A，然后自己拿着$max(A_{max}，B_{max})$。

综上所述，他们俩虽说玩了k轮游戏，实际上只玩了几轮游戏啊？2轮！！如果k是个奇数，就只玩了1轮，如果k是个偶数，就只玩了2轮，咱们代码是不是就可以改一改了。

```cpp
#include <bits/stdc++.h>
using namespace std;
int a[1005], b[1005], n, m, k, p; //p就相当于真真正正只玩了多少轮
long long sum;
int main() {
    cin >> n >> m >> k;
    for (int i = 0; i < n; i++) cin >> a[i];
    for (int j = 0; j < m; j++) cin >> b[j];
    sort(a, a + n);
    sort(b, b + m);
    if (k % 2 == 1) {
        //如果K == 1相当于只玩了1轮
        p = 1;
    } else {
        //否则相当于只玩了2轮
        p = 2;
    }
    for (int i = 1; i <= p; i++) {//虽说游戏玩了K轮，相当于只玩了P轮
        if (i % 2 == 1) {
            if (a[0] < b[m - 1]) {
                swap(a[0], b[m - 1]);
            }
        }
        if (i % 2  == 0) {
            if (b[0] < a[n - 1]) {
                swap(b[0], a[n - 1]);
            }
        }
        sort(a, a + n);
        sort(b, b + m);
    }
    for (int i = 0 ; i < n; i++) {
        sum += a[i];
    }
    cout << sum << endl;
    return 0;
}
```

> 时间复杂度$O(nlogn)$，每一轮需要对a,b排序一次，时间复杂度为$O(nlogn)$。一共最多进行2轮，忽略不计K
>
> 空间复杂度为$O(n+m)$，n是A的长度，m是B的长度，我们需要存储他们的分。

[U393235]奇数区间 https://www.luogu.com.cn/problem/U393235

期末考试T4

题目描述

给定一个长度为 n 的数列：$a_1, a_2, \cdots a_n$，如果其中一段**连续的**子序列 $a_i, a_{i+1}, \cdots a_j (i \leq j)$中，奇数比偶数多，我们就称这个区间 $[i, j]$ 是**奇数区间。**

你能求出数列中总共有多少个**奇数区间**吗?

输入格式

第一行包含一个整数 n 。

第二行包含 n 个整数 a_i

输出格式

输出一个整数，代表**奇数区间**的数目。

样例 #1

样例输入 #1

```
7
1 3 2 4 5 6 7
```

样例输出 #1

```
9
```

提示

【样例解释】

奇数区间共有9个，分别为：**[1，1]，[1，2]，[1，3]，[1，5]，[1，7]，[2，2]，[5，5]，[5，7]，[7，7]。**

【数据范围】

对于所有测试数据有：$1 \leq n \leq 10^6$，$10^{-9} \leq a_i \leq 10^9$。

测试点编号	$n \leq$	特殊性质A	特殊性质B	特殊性质C
1,2	10^3	否	否	否
3	10^3	否	否	否
4	10^5	否	是	否
5	10^5	否	否	是
6	10^5	否	否	否
7	10^6	是	否	否
8	10^6	否	是	否
9	10^6	否	否	是
10	10^6	否	否	否

特殊性质A：输入的 n 个整数全是偶数。

特殊性质B：输入的 n 个整数全是奇数。

特殊性质C：输入的 n 个整数中，奇数偶数均匀分布，不会有任何一个区间中奇数个数减偶数个数或者偶数个数减奇数个数的值大于10。

这道题目非常非常难，不要求各位同学能够完全掌握这道题目正确的解法。大家做个了解就好。

要求的是奇数的个数比偶数多的子数组的个数，那我们该如何表示一个区间内，奇数的个数比偶数多呢？大家有没有想到前缀和，我用一个前缀和数组来记录奇数的个数，用一个前缀和数组来记录偶数的个数，这样如果一段区间内，奇数的个数大于偶数，就对答案加一。各位同学是这样想的吗？还能不能更优一点呢？可不可以只用一个前缀和数组呢？在这，老师教给大家一个技巧，我们把奇数看成是1，对答案有正贡献，把偶数看成是-1，对答案有负贡献，那如果一段区间内的奇数的个数大于偶数，所以1的个数应该大于-1的个数，这段区间的和理应 >0。反之，当一段区间的和是≤ 0的，则说明这是个不符合条件的区间。

那我们记$s[i]$表示子数组$[0..i]$的和。在以下，我们称奇数的个数比偶数多多少个叫做"和"。

记$s[j]$ 表示$[0..j]$的和，记$s[k]$表示$[0..k]$的和，其中满足$0 \leq k < j$ 那么存在 $s[j]-s[k-1]>0$ 则表示$[k..j]$这个区间内奇数的个数大于偶数的个数，这是一个合法的区间。

我们再建立一个前缀和的哈希表$(map)mp$，初始的，我们认为$s[-1]=0$，其含义为空数组的奇数比偶数多0个，同样的，$mp[0]=1$，奇数比偶数多0个的情形，在空数组中已经出现1次。当我们每次遍历到$a[i]$时，令$mp[x]=$满足$0 \leq k < i$且$s[k]=x$的个数，这样，以i结尾的，奇数比偶数多的子数组个数就是cnt数组中，所有小于$s[i]$的下标和(记为cnt)，而这个值由于当前的一项i只与前一项$i-1$有关，所以我们可以**动态维护**。

- 如果$a[i]$是个偶数，也就是$a[i]\%2==0$，则$s[i]=s[i-1]-1$，$cnt=cnt-mp[s[i]]$
- 如果$a[i]$是个奇数，也就是$a[i]\%2\neq 0$，则 $s[i]=s[i-1]+1$，$cnt=cnt+mp[s[i]-1]$

最后更新一下$mp[s[i]]$，对它加1，答案$ans+=cnt$

如果同学们没有明白cnt是干嘛的，我们不妨列个表格。

	0	1	2	3	4	5	6
$a[i]$	1	3	2	4	5	6	7
$s[i]$	1	2	1	0	1	0	1
cnt	1	2	1	0	2	0	3
ans	1	3	4	4	6	6	9
mp	$mp[1]=1$	$mp[2]=1$	$mp[1]=2$	$mp[0]=2$	$mp[1]=3$	$mp[0]=3$	$mp[1]=4$

那么我们可以写出如下代码：

```cpp
#include <bits/stdc++.h>
using namespace std;
unordered_map<long long, int> mp;
long long sum[1000010];
long long cnt, ans;
int main() {
    int n, x;
    cin >> n;
    mp[0] = 1;
    for (int i = 1; i <= n; i++) {
        cin >> x;
        if (x % 2 == 0) { //如果当前是个偶数
            sum[i] = sum[i - 1] - 1; //前缀和数组 - 1
            cnt -= mp[sum[i]];       //cnt减少sum[i]出现的次数
```

```
        } else { //如果当前是个奇数
            sum[i] = sum[i - 1] + 1; //前缀和数组 + 1
            cnt += mp[sum[i] - 1];   //cnt增加sum[i] - 1 出现的次数。
        }
        ans += cnt;     //对答案累加
        mp[sum[i]]++;   //把当前sum[i]这个前缀和出现的次数++;
    }
    cout << ans << endl;
    return 0;
}
```

> 时间复杂度$O(n)$，n是数组的长度，需要对数组进行一次遍历，每次加入哈希表的时间复杂度为$O(1)$
>
> 空间复杂度$O(m+C)$，m是前缀和数组的长度，C是一个常数，表示哈希表map需要开辟的空间。

那我们不难发现，前缀数组$sum[i]$有且仅与$sum[i-1]$有关， 可以对它**动态维护**

```
#include <bits/stdc++.h>
using namespace std;
unordered_map<long long, int> mp;
long long sum, cnt, ans;
int main() {
    int n, x;
    cin >> n;
    mp[0] = 1;
    for (int i = 1; i <= n; i++) {
        cin >> x;
        if (x % 2 == 0) { //如果当前是个偶数
            sum--; //前缀和 - 1
            cnt -= mp[sum];       //cnt减少sum[i]出现的次数
        } else { //如果当前是个奇数
            sum++; //前缀和 + 1
            cnt += mp[sum - 1];   //cnt增加sum[i] - 1 出现的次数。
        }
        ans += cnt;     //对答案累加
        mp[sum]++;   //把当前sum这个前缀和出现的次数++;
    }
    cout << ans << endl;
    return 0;
}
```

> 时间复杂度$O(n)$，n是数组的长度，需要对数组进行一次遍历，每次加入哈希表的时间复杂度为$O(1)$
>
> 空间复杂度$O(C)$，C是一个常数，表示哈希表map需要开辟的空间。除此之外，我们只需要使用常数个变量