信息学奥赛笔记10

2024-01-20期末考试试题

T1猫猫序列

题目描述

有 n 个盒子对齐放在一排，里面可能藏着一只猫，我们用序列 $b_1, b_2, ..., b_n$ 来描述盒子中有没有猫，如果 b_i 的值为 1 ，表示第 i 个盒子中有一只猫，否则如果 b_i 的值为 0 ，表示第 i 个盒子中没有猫。

你可以**选做**下面三种操作：

1、给一个空盒子里放一只猫（假设第 i 个盒子是空盒子， b_i 的值从 0 变为 1。）

2、将一个有猫的盒子里的猫移出（假设第 i 个盒子有猫，b_i 的值从 1 变为 0。）

3、将一个有猫的盒子里的猫放到一个空盒子中（假设第 i 个盒子有猫，第 j 个盒子是空盒子，将第 i 个盒子里的猫放到第 j 个盒子中，b_i 的值从 1 变为 0，b_j 的值从 0 变为 1。）

对于每个盒子中猫的状态，我们有一个期望序列，$a_1, a_2, ..., a_n$，现在要求你对实际猫的状态序列 $b_1, b_2, ..., b_n$ 通过选择以上三种操作，实现经过 m 次操作后，b 序列等于 a 序列，求**最少的**操作次数。

输入格式

第一行包含一个正整数 n ，表示一排盒子的数量，或者说序列长度。

第二行包含 n 个正整数 $a_1, a_2, ..., a_n$，表示最终对猫在盒子中状态的期望序列。

第三行包含 n 个正整数 $b_1, b_2, ..., b_n$,表示猫在盒子中最初的状态序列。

题目保证输入的序列 a_i, b_i 的值均为 0 或 1。

输出格式

输出一行，仅包含一个正整数，表示做的最少操作次数。

样例 #1

样例输入 #1

```
5
00001
10010
```

样例输出 #1

```
2
```

提示

【样例 1 解释】

将猫从第一个盒子移动到第五个盒子，然后从第四个盒子中移出猫。

【数据范围】

对于 30 的数据，有$1 \le n \le 10^3$；

对于 100 的数据，有$1 \le n \le 10^7$。

算法思路

> 读完题后我们可以得知题目给了我们两个01串(只包含0和1的字符串)，让我们把第二个01串变成第一个01串，允许我们把串中的1移动位置，或者0变1，1变0。我们需要变成的那个期望串叫a串，我们把要操作的字符串叫做b串，那也就是说，a串和b串在位置i上会有下列的四种情况

a_i是0, b_i是0，不需要任何操作

a_i是1，b_i是1，不需要任何操作

a_i是0，b_i是1，说明在第i个位置上原本没有猫，但是现在我们要操作的字符串上有猫，我们可以对这个猫做**清除**或者**移动**

a_i是1，b_i是0，说明在第i个位置上需要有只猫 ，但是我们先要操作的字符串上没有猫，我们需要从别的地方**移来**或者**新增**

> 那对这道题目分析完毕，我们可以发现一件事情，把猫从一个盒子i移动到另外一个盒子j相当于：删除i上的猫，为j上新增一个猫，两步，既然我们发现移动猫相当于2步增删操作，那我们就可以尽可能的利用这些猫来做移动，比如在刚刚我们所说到的第③种情况中，这代表我们多了一个可以随时移动的猫，那我们可以把字符串当中的a_i是0，b_i是1 统计到$cnt1$， a_i是1，b_i是0 统计到$cnt2$。

那根据刚刚对四种情况的描述，我们可以发现。如果$cnt1 >= cnt2$：

　　说明我们可以调度使用的猫有$cnt1$只，数量多于我们需求的$cnt2$只猫，那我们需要从这$cnt1$只猫当中选出$cnt2$ 只猫移动到这$cnt2$只猫应有的位置上，再把剩下$cnt1 - cnt2$只猫删去，一共需要执行$cnt1 - cnt2 + cnt2 = cnt1$ 次操作

如果$cnt1 < cnt2$ 呢?

　　那我们应该把这$cnt1$只猫全部移动到$cnt1$只猫应有的位置，还剩下$cnt2 - cnt1$只猫需要我们使用添加的形式把猫加入。总共需要执行$cnt2 - cnt1 + cnt1 = cnt2$ 次操作

那我们不难发现，答案就是$max(cnt1, cnt2)$ 综上所述，我们可以写出如下代码：

```
#include <bits/stdc++.h>
using namespace std;
int n, cnt1, cnt2;
string a, b;
int main() {
    cin >> n >> a >> b;
    for (int i = 0; i < n; i++) {
        if (a[i] == '0' && b[i] == '1') cnt1++;
        if (a[i] == '1' && b[i] == '0') cnt2++;
    }
    cout << max(cnt1, cnt2) << endl;
    return 0;
}
```

> 时间复杂度为O(n)，n表示字符串的长度，我们需要对2个字符串进行1遍扫描
>
> 空间复杂度为O(n)，n表示字符串的长度，我们需要存储2个字符串

T2特权世界

题目背景

D 星球是一个特权世界，里面每个 D 球人自出生起就自带一个属性值，属性值越高的 D 球人无论做什么事都优先于其他人。可想而知， D 星球充斥着属性值的对决。

题目描述

最近，D 星球 A 市要举办一场大型活动，D 各市的人陆续赶来，主办方会对到来的属性值最高的 m 个人做登记，若后续有更高属性的人来参加活动，主办方就会告知已登记的人中最低属性的人“您已被挤掉资格”并给予被挤掉的人补偿（同属性值不会被挤掉）。为了鼓励 D 球人踊跃参加活动，同样的属性值，到达越早的人优先参加活动。
未到最后一刻，原本在榜的 m 个人都不知道自己会不会被后续的高属性人挤掉。

你是这次补偿品的购买者，请你根据每个人的到场时间，计算需要补偿多少人，并按从小到大的顺序输出最终可参加活动的 D 球人的属性值。

输入格式

第一行有两个正整数 n 和 m ，表示有 n 个人赶过来参加活动，最终能参加活动的人有 m 个。

第二行有 n 个正整数 $a_1, a_2, ..., a_n$，表示赶来参加活动的人的属性值。

输出格式

第一行一个正整数，表示要补偿的人数。

第二行 m 个正整数，表示最终可参加活动的人的属性值，每个属性值用空格隔开。

样例 #1

样例输入 #1

```
10 3
34 2 1 7 6 19 20 45 5 99999999
```

样例输出 #1

```
6
34 45 99999999
```

提示

【数据范围】

对于 30% 的数，$1 \le m \le n \le 10^3, 1 \le a_i \le 10^9$。

对于 100% 的数据，$1 \le m \le n \le 5 \times 10^5, 1 \le a_i \le 10^{18}$。

算法思路

> 如果读完题后你的第一反应是排序，先抽自己一下，要注意啊题目中描述的是陆续到来的人，什么叫做陆续？这些人的顺序是不能改变的！！不能改变的！！不能改变的！！那么，既然人来的顺序不能改变，你又要获取到这些人中的最大的m个人，那我们就要想到要使用一种能够按照顺序的，能够直接帮助我们排序的STL，是什么？优先级队列！

这种类型的题目一律统称为叫$TOP-K$问题，也就是说，我们需要动态的，获取到一个数组里的前k大的值或者前k小的值，很明显这道题目是$TOP-M$，那我们来梳理一下这道题目的逻辑。

优先级队列提供给我们四个常用的函数

push	pop	top	size
丢数进去	扣数出来	获取目前的队头元素	求队列现在有几个元素

那么，既然我们要控制队列里的元素是前m个最大值，那就想，你应该抛弃掉大的还是小的？小的对吧，结果小的你不想要，你想丢掉，所以要出队的时候出**小值**。默认的优先级队列是出**最大值**，所以在写优先级队列的时候，就要改写成

```
prority_queue<类型，vector<类型>，greater<类型> >
```

　　这种定义规则，这道题的每个数最大高达10^{18}所以这里的类型就是$longlong$

那么，我们该如何控制队列里一直存的有m个最大的数呢？

两种情况:

如果当前队列里的元素个数比m要小，怎么办，无条件入队，最起码得让队列里先得有m个元素对吧。

那如果元素个数已经达到了m，就不能再无条件入队了，为了维持队列中有m个元素，此时如果进入一个，就得再出一个，此时就有两种情况，如果当前我想插进去的元素比队列里面我保存的最小的元素还小，那还有没有必要让他进去了，没必要了对吧，那否则呢，就说明，当前这个元素可以替换掉队列里面最小的那个值，此时就会发生题目当中说到的**"挤掉"**,我们要统计的就是**"挤掉"**一共发生了多少次。

代码如下:

```cpp
#include <bits/stdc++.h>
using namespace std;
priority_queue<long long, vector<long long>, greater<long long> > pq; //定义一个每次出队是最小值的long long的优先级队列
int n, m, ans = 0;
long long x;
int main() {
    cin >> n >> m;
    for (int i = 0; i < n; i++) {
        cin >> x;
        if (pq.size() < m) {//如果当前队列的元素是小于m的
            pq.push(x);//无条件入队
        }
        else if (x > pq.top()) {//否则，说明当前队列的元素是等于m的，那就要考虑当前这个x有没有入队的资格了
            pq.pop();
            pq.push(x);
            ans++;
        }
    }
    cout << ans << endl;
    while (!pq.empty()) {//只要队列里还有元素，就输出，这里也可以写成pq.size()或者m--或者for(1到m)
        cout << pq.top() << " ";
        pq.pop();
    }
    return 0;
}
```

> 时间复杂度为$O(nlogn)$，对于每次入队操作，时间复杂度为$O(logn)$，最坏的情况需要执行n遍入队操作
>
> 空间复杂度为$O(m)$，队列中存储的数最多为$m+1$个

T3小L的糖果

题目背景

元旦节到了，小L所在的学校举办了联欢会，在联欢会的项目中胜利的同学可以获得一堆糖果。

题目描述

小L负责糖果的分发。一开始桌子上有共n（n为奇数）堆糖果，糖果盒中还有k枚糖果，他可以为任意一堆新增若干枚糖果，但是不可以把其中一堆移动到另外一堆。为了奖励小L为联欢会布置的辛苦，老师决定让他选择一堆糖果作为奖品，小L比较纠结，想要获得不多不少的糖果，他决定拿走这些糖果**中位数**的那一堆，但是小L也是个贪婪的孩子，他希望拿走的糖果尽可能多，现在他来求助于你，你可以帮帮他吗？

输入格式

第一行两个正整数分别为n和k，表示一开始桌子上糖果的堆数，和糖果盒中糖果的个数，保证n是一个奇数。

接下来一行有n个正整数，第i个整数a_i表示第i堆糖果的初始个数。

输出格式

一个整数，表示小L最多可以获得多少块糖果

样例 #1

样例输入 #1

```
3 2
1 3 5
```

样例输出 #1

```
5
```

样例 #2

样例输入 #2

```
7 7
4 1 2 4 3 4 4
```

样例输出 #2

```
5
```

提示

对于 30% 的数据，$1 \leq n \leq 100$，$1 \leq a_i$，$k \leq 1000$。

对于 70% 的数据，$1 \leq n \leq 1000$，$1 \leq a_i$，$k \leq 1 \times 10^6$。

对于 100% 的数据，保证n是一个奇数，$1 \leq n \leq 2 \times 10^5$，$1 \leq a_i$，$k \leq 1 \times 10^{12}$。

说明

中位数表示一个数组**排序后**位于**中间**的值，例如数组$[2,6,4,7,5]$排序后为$[2,4,5,6,7]$，中位数为5。

在**样例#1**中，小L可以把2块糖果都放到第2堆糖果，把糖果变成$[1,5,5]$。他最多能获得的糖果为5。可以证明没有比获得5枚糖果更多的方案。

> 什么叫做中位数，是一个数组中**排序后**位于中间的值，通过题目当中的第二个样例我们可以发现，没法直观的看出什么是中位数，所以需要对数据进行排序。但是，根据同学们的用法习惯不同，我们可以发现中位数取得的情况也不一样。
>
> 如果同学喜欢把数组开辟为$[1..n]$ 那么中位数应该是$a[(n+1)/2]$

> 如果同学喜欢把数组开辟为$[0..n-1]$，那么中位数应该是$a[n/2]$，本题解中以下均以数组开辟为$[0..n-1]$的情况来进行说明。
>
> 题目告诉我们可以进行k次操作。什么被称之为1次操作呢？那就是将1个数$+1$。
>
> 那题目希望我们在执行完k次操作后，使得这组数当中的中位数最大，那大家可以思考一下对于第二个样例，我们初始的中位数为4，我该怎么把中位数的值提高到5？那是不是我们把数组变成$[1, 2, 3, 5, 5, 5, 5]$，也就是说，只提高中位数是不行的，我们需要把中位数往后的所有数全体提高，才会导致中位数能够提高，在这里我们又发现了，对于比中位数要小的值来说，提高他们有这个必要吗？没有，纯粹是浪费操作次数的。既然想要提高中位数，所以我们应该把中位数往后的数**尽可能的平均的提高。** 同学们可以好好地思考一下这句话的含义，为什么我们要尽可能平均的吧中位数右边那侧的值提高，因为如果你只提高中位数的话，它变成最大的值，被移动到了数组的末尾，那么刚刚和中位数相等的值就又变成了中位数，你继续提高那个极大的值，中位数不受影响，所以你应该将他们都提高到某一值才能保证中位数提高。到底应该提高到几呢？

> 那我们进一步深入思考一下，假设，我想把这组数据的中位数提高到**104** 需要几次操作呢？ 答案是$(104-4)*4=400$ 次，那么样例中给定我们的操作次数为7次操作，那同学们可以想一想，既然提高到**104**已经不可能了，我还有没有必要去思考能不能把中位数提高到**204**？ 是不是更没戏？
>
> 那同学们再想一想，我想把这组数据的中位数提高到 **5**，需要多少步操作？ **4**步。在给定我们的操作范围内，那我有没有必要去考虑把中位数提高到 **4, 3, 2 ...**，是不是没有必要所以我们通过分析发现了这道题目的单调性，如果我们猜测一个目标中位数x，发现这个中位数需要花费的操作大于k，那就意味着大于x的答案更不可能，更大于k， 那如果我们猜测的x 需要花费的操作小于等于k，说明它符合，比x小的答案应该都符合，更符合了，这就是具备了**二分贪心的单调性**，所以这道题目应该采用二分答案的做法，猜什么？猜**这组数据改为中位数为x的数组需要多少步操作**，看这个操作的步骤和k 比较的结果进行二分。

代码如下（关于这道题目最大的痛点就是：~~五年OI一场空，不开long long见祖宗~~，一定要养成开long long的好习惯）：

```cpp
#include <bits/stdc++.h>
using namespace std;
long long n, k, l, r;
long long a[200010];//注意题目中数组的范围为2e5

bool check(long long x) {
    long long cnt = 0;//cnt统计如果中位数为x的话需要多少步操作次数，别忘记开long long！！！
    for (int i = n / 2; i < n; i++) {//循环从中位数开始，因为小于中位数的值咱不需要管他
        if (a[i] >= x) break;//如果当前这个元素已经大于等于x了，那说明它符合要求了已经，后面的数是更大的数，他们更符合要求，直接break
        cnt += x - a[i];//x是目标数，a[i]是当前数，目标数 - 当前数 = 操作次数， 雷加操作次数
    }
    return cnt <= k;//合法的操作是我们花费的操作步数小于给我们的操作步数
}

int main() {
    cin >> n >> k;
    for (int i = 0; i < n; i++) {
        cin >> a[i];
    }
    sort(a, a + n);
    l = 0, r = a[n / 2] + k + 1; //注意，由于在二分的时候我们没法找到r初始值的那个值，所以答案右区间需要 + 1。
```

```
    while (r - l != 1) {//二分板子
        long long x = l + (r - l) / 2;//别忘记开long long
        if (check(x)) {
            l = x;
        } else {
            r = x;
        }
    }
    cout << l << endl;//左区间表示cnt <= k的合法区间，所以答案是慢慢变大的，取得是l
    return 0;
}
```

> 时间复杂度为$O(nlogC)$，对于每次猜答案，需要对数组扫描一遍，时间复杂度为$O(n)$，C表示元素的最大值 + k,最坏的情况下要执行$logC$次
>
> 空间复杂度为$O(n)$，n表示数组的长度