信息学奥赛笔记22

考试题目解析 | 市赛小高组题目解析

0505测试题

卡车

题目描述

S国的卡车有两个油箱，主油箱中有a加仑燃料，副油箱有b加仑燃料。

该卡车在一条笔直的马路上匀速行驶，每消耗1加仑燃料都可以行驶xkm，且每当主油箱消耗k加仑燃料时，如果副油箱有燃料，都将从副油箱转移1加仑燃料到主油箱。

求这辆卡车最大的行驶距离。

注意：从副油箱转移至主油箱并不是一个连续的的行为，这一事件在每消耗k加仑燃料时，突然且瞬间发生。

输入格式

一行四个整数，a，b，x，k，符合题目描述。

输出格式

一个整数，表示所求答案。

样例 #1

样例输入 #1

```
5 10 10 5
```

样例输出 #1

```
60
```

提示

对于50%的数据，有$1 \leq a, b \leq 10^4$，$1 \leq k, x \leq 10^3$。

对于100%的数据，有$1 \leq a, b \leq 10^9$，$1 \leq k, x \leq 10^9$。

样例解释：

用掉5加仑燃料时，副油箱向主油箱传递1加仑燃料，卡车共行驶$50km$，再次消耗1加仑燃料，卡车再行驶$10km$，共计$60km$。

思路解析

> 这道题也没什么思路，就是纯粹的模拟，算出主油箱的可行驶公里数，主油箱每消耗k加仑油后副油箱向主油箱转移，再次计算主油箱公里数，循环往复即可，这道题不能拿满分就是纯粹是代码能力不行。和思维，和算法都没有关系。

```cpp
#include <bits/stdc++.h>
using namespace std;
int main() {
    long long a, b, x, k, ans = 0, cnt = 0;
    cin >> a >> b >> x >> k;
    while (a) {
        ans += a * x;  // 计算主油箱
        cnt += a;      // 计算主油箱累计消耗
        a = min(cnt / k, b);  // 计算副油箱向主油箱转移量，不能超过副油箱的已有油量
        cnt %= k;  // 主油箱累计消耗记得取剩余
        b -= a;    // 副油箱油量减少
    }
    cout << ans;  // 输出答案
    return 0;
}
```

能量牌

题目描述

小A和小B在玩一款特殊的游戏。这个游戏的规则如下，小A有n张卡牌，每张牌上有一个点数，表示卡牌的能量。

小B作为防守方，拿出了m张卡牌防守，每张卡牌拥有一个护盾量，当能量牌大于护盾量的时候，这张牌就会被击碎。

轮到小A操作了，假设他是一个绝顶聪明的人，请你帮助小A计算一下，这一回合内他最多可以击碎小B多少张卡牌。

输入格式

第一行2个整数n，m，表示小A派出进攻牌的数量和小B派出防守牌的数量。

第二行共n个整数，每个整数a_i表示小A第i张卡牌的能量。

第三行共m个整数，每个整数b_i表示小B第i张卡牌的护盾量。

输出格式

一行一个整数，表示小A最多能击碎小B多少张卡牌

样例 #1

样例输入 #1

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

样例输出 #1

```
2
```

提示

对于30%的数据，有$1 \leq n$，$m \leq 10^3$，$1 \leq a_i$，$b_i \leq 10^3$。

对于100%的数据，有$1 \leq n$，$m \leq 2 * 10^5$，$1 \leq a_i$，$b_i \leq 10^9$。

样例解释：

第1轮，小A用第3张牌击碎小B第1张牌，小A用第4张牌击碎小B第3张牌，最多可以击碎他两张牌。

思路解析

> 对于a的第i张牌a_i来说要想物尽其用，我们应该用a_i击碎b尽可能防御较高的牌，这样我们才能尽可能去使用a攻击数值较低的牌，那如果a_i已经无法击碎b_j了，那么对于所有大于b_j的牌，所有小于a_i的牌，都是不可能击碎的，既然如此，也就没有循环搜索的意义了，所以可以使用双指针同时遍历两个数组来做这道题。

```cpp
#include <bits/stdc++.h>
using namespace std;
int main() {
    int n, m;
    cin >> n >> m;
    vector<int> a(n), b(m);
    for (int i = 0; i < n; i++) cin >> a[i];
    for (int j = 0; j < m; j++) cin >> b[j];
    sort(a.begin(), a.end(), greater<int>()); // 对两个数组都进行从大到小排序
    sort(b.begin(), b.end(), greater<int>());
    long long ans = 0;
    for (int i = 0, j = 0; i < n && j < m; i++, j++) {  // 用i遍历a数组，j遍历b数组
        while (j < m && a[i] < b[j]) j++;  // 如果a[i]无法击碎b[j]，我们就找一个a[i]尽可能可以击碎的牌
        if (j == m) break;  // 特殊判断，如果a[i]无法击碎任何一张b的牌，意味着小于a[i]的牌也不再有意义，直接结束循环。
        ans++;
    }
    cout << ans;
    return 0;
}
```

质数乐园

题目描述

小L在一个数学游乐场，他在玩一个打靶游戏，在他的面前一共摆放了n个标有数字的气球，他每击中一个气球，必须要报出离气球上数字最近的质数才能得分，现在依次把他击中气球上的数字给你，请你来帮他完成这个报数字的任务。

输入格式

第一行一个整数n，表示气球的个数。

第二行n个整数，每个整数a_i表示第i个气球上的数字。

输出格式

一行n个整数，对应每一个气球命中时应该报出的数字。如果这个数离他最近的质数不只一个，请你输出较小的那一个。

样例 #1

样例输入 #1

```
3
6 9 13
```

样例输出 #1

```
5 7 13
```

提示

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

对于50%的数据，有$1 \leq n \leq 10^4$，$1 \leq a_i \leq 10^3$。

对于80%的数据，有$1 \leq n \leq 10^5$，$1 \leq a_i \leq 1000$。

对于100%的数据，有$1 \leq n \leq 10^5$，$1 \leq a_i \leq 5 * 10^6$。

思路分析

> 首先通过欧拉筛获取所有质数。在获取到的质数序列中，我们要获取距离当前这个数最近的质数，我们可以对整个质数序列进行二分查找，找到刚好大于等于当前这个数$a[i]$的位置j，那么距离$a[i]$最近的质数只可能是$b[j]$，$b[j-1]$，取距离最近的输出即可。

```cpp
#include <bits/stdc++.h>
using namespace std;
int st[5000001];
int main() {
    memset(st, 0, sizeof(st));
    int n, x;
    cin >> n;
    vector<int> prime;
    for (int i = 2; i <= 5e6; i++) {
        if (!st[i]) prime.push_back(i);
        for (int j = 0; j < prime.size() && i * prime[j] <= 5e6; j++) {
            st[i * prime[j]] = 1;
            if (i % prime[j] == 0) break;
        }
    } // 欧拉筛获取质数
    for (int i = 0; i < n; i++) {
        cin >> x;
        int y = upper_bound(prime.begin(), prime.end(), x) - prime.begin();
 // 二分查找一个刚好大于x的质数的位置y
```

```
        if (!y) {  // 如果y == 0,特殊判断，相当于x == 1的情况，刚好比1大的质数是第0号质数，也就是2，直接输出2，否则prime[y - 1]下标越界
            cout << 2 << " ";
            continue;
        }
        if (abs(prime[y] - x) < abs(prime[y - 1] - x)) {  // 寻找一个距离最近的质数
            cout << prime[y] << " ";
        } else cout << prime[y - 1] << " ";
    }
    return 0;
}
```

最大01串奇数和

题目描述

给定你两个`01字符串` `a, b`，它们代表着两个二进制的整数。现在你可以对每个字符串进行重新排列后对两个这两个二进制整数字符串进行求和，在和为一个**奇数**的情况下，输出最大可能的和。

`01字符串`指的是只包含字符`0`和字符`1`的字符串，且保证输入数据有解。

注意：你只能使用输入的两个字符串进行重排，不能对字符串整数补前导0或者删除前导0，例如：001可以重排为010，但是不能把它重排为00010。

输入格式

两行，每行一个字符串，仅包含'0'和'1'。

输出格式

一个字符串，表示答案。

样例 #1

样例输入 #1

```
1010
0100
```

样例输出 #1

```
10001
```

提示

设$l1$，$l2$为字符串a，b的长度。

对于10%的数据，有$1 \leq l1, l2 \leq 10$。

对于40%的数据，有$1 \leq l1$，$l2 \leq 10^4$。

对于100%的数据，有$1 \leq l1$，$l2 \leq 2 * 10^5$。

对于额外10%的数据：字符串a，b各只含有一个'1'。

对于额外10%的数据：有$l1 = l2$。

对于额外10%的数据，字符串a, b种含有的1均**不**大于字符串长度的一半。

保证字符串中仅含有'0'和'1'

样例解释：

第一个数重排为1001，第二个数重排为1000，结果为10001，是个奇数，可以证明，没有比10001更大的答案。

思路分析

> 如果要保证两个二进制数的和是一个奇数，则结果的最后一位必须要是1。那为了尽可能让结果变大，我们就得把原本的两个二进制数的1都调到高位，最终我们可以通过交换其中一个字符串的一个1到末尾的方式来比较哪一个答案会更大，计算过程需要使用二进制的高精度运算。

```cpp
#include <bits/stdc++.h>
using namespace std;
string add(string s1, string s2) {  // 该函数的作用是对两个二进制的字符串进行高精度加法
    string ans;
    int i = s1.size() - 1, j = s2.size() - 1, x = 0;
    while(i >= 0 || j >= 0) {
        int a = i >= 0 ? s1[i--] - '0' : 0;
        int b = j >= 0 ? s2[j--] - '0' : 0;
        x += a + b;
        ans += x % 2 + '0';
        x /= 2;
    }
    if (x) ans += "1";
    reverse(ans.begin(),ans.end());
    return ans;
}
int main() {
    string a, b, ans;
    cin >> a >> b;
    sort(a.begin(), a.end(), greater<char>());
    sort(b.begin(), b.end(), greater<char>());  // 将两个字符串进行排序，按照从大到小，这样1都跑到高位去了
    int l1 = a.size(), l2 = b.size(), l = 0, r = 0;
    while (l < l1 && a[l] == '1') l++;
    while (r < l2 && b[r] == '1') r++;  // l, r定位到每个字符串最后一个1
    string ans1 = "", ans2 = "";
    if (l && l != l1) {
        string t = a;
        swap(t[l - 1], t[l1 - 1]);  // 尝试对1串把最后一个1换到末尾
        ans1 = add(t, b);
    } else if (l) ans1 = add(a, b); // 特殊情况判断，如果第一个串没有1，直接将两个字符串相加
    if (r && r != l2) {
        string t = b;
        swap(t[r - 1], t[l2 - 1]);  // 尝试对2串把最后一个1换到末尾
        ans2 = add(a, t);
```

```
    } else if (r) ans2 = add(a, b); // 特殊情况判断，如果第二个串没有1，直接将两个字符串相加
    if (ans1.size() > ans2.size()) cout << ans1; // 输出结果时，可以先看哪个字符串更长，长的一定更大，当字符串一样大时，可以直接用max比较
    else if (ans2.size() > ans1.size()) cout << ans2;
    else cout << max(ans1, ans2);
    return 0;
}
```

小高组市赛考试

交替字符串

题目描述

给定一个只包含两种字符的字符串`s`，如果`s`的一个子串中**不存在**两个**相邻的**字符相同的情况，则认为这是一个**交替子字符串**。

需要注意的是，两个子串起始位置不同，终止位置不同，则认为他们是不同的子串。

求字符串`s`的交替子字符串的个数。

输入格式

一行，一个字符串`s`，仅包含两种字符。

输出格式

一个整数，表示字符串`s`的交替子字符串的个数。

样例 #1

样例输入 #1

```
abbb
```

样例输出 #1

```
5
```

提示

对于30%的数据，有$1 \leq s.size() \leq 10^3$。

对于100%的数据，有$1 \leq s.size() \leq 10^6$。

保证输入数据只出现大写字母和小写字母且字符串中有且仅有两种字符。

思路分析

> 对于一个本身是交替字符串的字符串来说，例如abab，那么它的每一个子串都是交替子字符串，对于一个有相邻字符的情况来说，例如abbabb，那么从相邻字符开始，当前这个字符必然不会再对之前的起点到当前的终点产生贡献，也就是说，如果两个字符相邻，当前这个重复的字符只能作为新的起点，不再能接续先前的串了。所以我们可以求出每一个字符对答案的贡献。设当前位置为i，字符串起点为j，当前的字符一共能产生$i-j+1$个贡献。

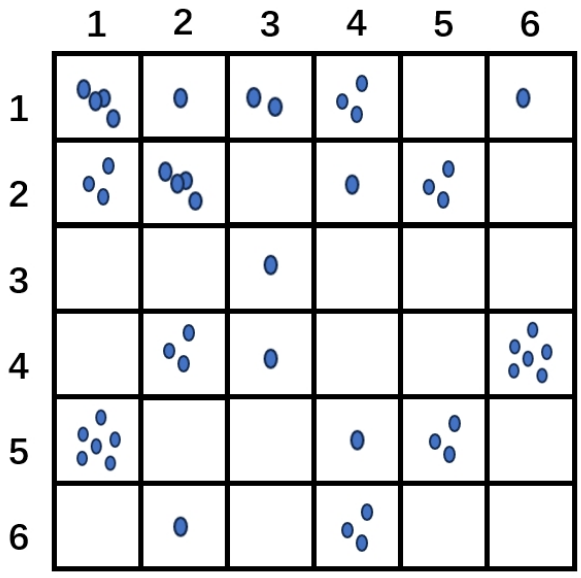
```cpp
#include <bits/stdc++.h>
using namespace std;
string s;
long long ans;
int main() {
    cin >> s;
    int n = s.size();
    for (int i = 0, j = 0; i < n; i++) {
        if (i && s[i] == s[i - 1]) j = i;  // 如果当前字符与相邻相等，则以当前这个作为起点
        ans += i - j + 1;    // 计算当前这个字符的贡献，并累加进答案中
    }
    cout << ans;
    return 0;
}
```

倒豆子

题目背景

“种豆得豆，种瓜得瓜”，豆子本从藤蔓上长出来，结果后可以将豆子种在地里，来年又会长出藤蔓结豆子。因此，豆子丰收时，除了收获食用售卖，还应该留好一定量的豆子作为种子。

题目描述



如图有一个正方形置物盘，置物盘上有 $n \times n$ 小格子，每个小格子里有数量不等的豆子。对于整个置物盘，我们可以做以下两种**操作**之一：

操作一： 选择置物盘中一行小格子，将这一行小格子中的豆子倒到相邻行中；再选择置物盘中一列小格子，将这一列小格子中的豆子倒到相邻列中。

操作二： 将整个置物盘沿顺时针转动一圈。

假设需要做 m 次操作，请你选择**最佳的**操作过程，使得做完操作后，将其中的一个格子里的豆子拿出来作为种子时，种子的数量**最多**。

输入格式

总共 $n+2$ 行。

第 1 行一个正整数 n，代表置物盘上的行数和列数。

第 2 到 $n+1$ 行每行 n 个非负整数 $a_{i,j}$，代表置物盘上第 i 行第 j 列的小格子里有 $a_{i,j}$ 个豆子。

第 $n+2$ 行一个正整数 m，代表操作的次数。

输出格式

一个正整数，代表最终拿出来的豆子的数量。

样例 #1

样例输入 #1

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

样例输出 #1

```
10
```

样例 #2

样例输入 #2

```
6
4 1 2 3 0 1
3 4 0 1 3 0
0 0 1 0 0 0
0 3 1 0 0 6
6 0 0 1 3 0
0 1 0 3 0 0
1
```

样例输出 #2

```
12
```

提示

【样例解释】

样例一：将第 1 行的豆子倒到第 2 行，再将第 1 列的豆子倒到第 2 列，此时第 2 行第 2 列的豆子数量是 10。

样例二：（该样例如题目中的图片所示）将第 1 行的豆子倒到第 2 行，再将第 1 列的豆子倒到第 2 列，此时第 2 行第 2 列的豆子数量是 12。

【数据范围】

对于 20% 的数据，有$2 \leq n \leq 10, 1 \leq m \leq 10, 0 \leq a_{i,j} \leq 100$。

对于 30% 的数据，有$2 \leq n \leq 100, 1 \leq m \leq 10, 0 \leq a_{i,j} \leq 100$。

对于 100% 的数据，有 $2 \leq n \leq 10^3, 1 \leq m \leq 10, 0 \leq a_{i,j} \leq 10^8$。

特殊数据：对于 20% 的数据， 保证 $m = 1$。

思路分析

> 数据是一个正方形，所以操作2是一个废的操作，不用去考虑操作2,。对于m = 1的情况，相当于移动一行，移动一列，也就是会导致一个2 * 2的正方形为最终的和，对于m = 2的情况，相当于移动2行，移动2列，最终会导致一个3 * 3的正方形为最终的和。那也就是说，对于任意一个m，就是求$(m + 1) * (m + 1)$的正方形最大的和，我们可以预处理前缀和来做这道题。

```cpp
#include <bits/stdc++.h>
using namespace std;
long long n, m, ans, x, sum[1001][1001];
int main() {
    cin >> n;
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            cin >> x;
            sum[i + 1][j + 1] = sum[i + 1][j] + sum[i][j + 1] - sum[i][j] + x;  // 求二维前缀和
        }
    }
    cin >> m;
    m = min(m, n - 1);  // 细节，m 和 n - 1不知道谁大，防止越界。
    for (int i = 1; i <= n - m; i++) {
        for (int j = 1; j <= n - m; j++) {
            ans = max(ans, sum[i + m][j + m] - sum[i + m][j - 1] - sum[i - 1][j + m] + sum[i - 1][j - 1]);  // 算二维前缀和
        }
    }
    cout << ans;
    return 0;
}
```

劳动最光荣

题目背景

勤劳是中华民族的传统美德，一年一度劳动节要到了，小L所在的学校在组织全校卫生清扫活动。

题目描述

小L所在的学校共有n名同学，每名同学在劳动节这一天被分配了基础劳动任务量，学校要求每名学生必须要达到m的劳动量才能获得“劳动之星”奖章。

但是很多同学以基础劳动量是不能够获得“劳动之星”的，所以学校允许自行组队以进行劳动量的分配，n名学生的其中一部分学生将以团体的形式组队活动，每名学生**至多**只能加入`1`个团队。此时团队中个人的劳动量将变为他们团队总计劳动量的**平均数**。

例如，将初始的劳动量记为`[4, 1, 3, 1]`，如果第`1`名同学和第`3`名同学自行组队，这两名同学的劳动总量为$4+3=7$，然后将$7/2=3.5$的劳动量平均分配给他们两人。因此，劳动量变为`[3.5, 1, 3.5, 1]`。

由于学生众多，信息量巨大，所以学校不知道进行了多少次组队，以及组队的对象都是谁，请你计算出在若干次组队后，获得“劳动之星”奖章的同学最大的可能数量。

输入格式

第一行包含两个整数n，m。分别表示学生的数量和获得“劳动之星”的劳动量。

第二行包含n个整数，表示全校学生的基础劳动量。

输出格式

一个整数，表示最大可能能获得“劳动之星”学生的数量。

样例 #1

样例输入 #1

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

样例输出 #1

```
2
```

样例 #2

样例输入 #2

```
3 7
9 4 9
```

样例输出 #2

```
3
```

提示

设第i名学生的劳动量为a_i：

对于10%的数据，有$1 \le n \le 10^3$，$1 \le m \le 10^4$，$1 \le a_i \le 10^3$。

对于100%的数据，有$1 \leq n \leq 10^5$，$1 \leq m \leq 10^9$，$1 \leq a_i \leq 10^9$。

样例解释 #1

> 按照题目中描述，重新分配劳动量为`[3.5, 1. 3,5, 1]`，最多可能有`2`名同学获得“劳动之星”。

样例解释 #2

> 所有的学生全体参与组队，重新分配劳动量后劳动量为[$7\frac{1}{3}$，$7\frac{1}{3}$，$7\frac{1}{3}$]，最多可能有`3`名同学获得“劳动之星”

思路分析

> 设$[a_1, a_2, \ldots a_n]$的和为sum，他们的平均值设为$x = sum/n$，再来一个值a_x，此时新的平均值变为$y = (sum + a_x)/(n + 1)$。也就是$(n * x + a_x)/(n + 1)$。
>
> 当$a_x > x$时，$y > x$。
>
> 当$a_x = x$时，$y = x$。
>
> 当$a_x < x$时，$y < x$。
>
> 通过这个结论我们可以得知一个事情，如果有一堆数，现在要给这一堆数再补进去一个数，这个数如果大于原本这堆数的平均值，导致平均值增大，如果等于这堆数原本的平均值，平均值不变，如果小于原本这堆数的平均值，平均值变小。
>
> 那将题目转化成数学模型之后，我们就是要求n个数最多有多少个数的平均值大于m。
>
> 所以应该采取的贪心策略为，尽可能的取大于m的数，这些数的平均值一定大于m，那么每有一个小于m的数加入进来，就会导致平均值的减少，那为了每次平均值减少的更少，我们应该取尽可能大的数，最终能得出最多有多少个数的平均值是大于m的，在做的时候，我们会发现，每次扫描一个小的数，如果剩余的数的平均值小于m，可以尝试将a_i从最终答案的数组中去除，也就是滚动前缀和，先计算出整个数组的和，每次看当前如果平均值比m要大，就意味着接下来的$n - i$个数是符合答案的，直接输出即可。如果不符合的话，就尝试把当前这个最小的a_i从答案中去除，再看看接下来的情况。

```cpp
#include <bits/stdc++.h>
using namespace std;
long long n, m, i, sum, a[100001];
int main() {
    cin >> n >> m;
    for (int i = 0; i < n; i++) {
        cin >> a[i];
        sum += a[i]; // 计算这n个数的和
    }
    sort(a, a + n);  // 对原始数组进行排序
    for (; i < n; i++) {
        if (sum >= m * (n - i)) break;  // 如果接下里的n - i个数是符合要求的，直接输出。
        sum -= a[i];  // 不符合要求，就去掉一个最小的数再看。
    }
    cout << n - i << endl;
    return 0;
}
```

吃饭

题目描述

小明饿了，所以小明要去吃饭。

桌子上摆了n道菜，第i道菜的编号是i，美味程度是a_i。

小明可以选择从任意一道菜开始吃（也可以不吃离场）。

由于餐厅的额外要求：假设小明吃的上一道菜的编号是x，那么小明吃的下一道菜的编号必须是x的倍数。

问：小明吃到的菜的美味程度之和最多是多少。

输入格式

第一行输入n。

第二行输入n个整数，表示$a_1, \ldots, a_n$。

输出格式

输出一个数字表示答案。

样例 #1

样例输入 #1

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

样例输出 #1

```
7
```

样例 #2

样例输入 #2

```
5
1 -1 4 7 5
```

样例输出 #2

```
8
```

提示

样例解释 #1

小明先吃1，再吃2，在吃4，可以得到$1+2+4=7$的美味程度。

样例解释2

小明先吃1，再吃4，可以得到$1+7=8$的美味程度。

更多样例见附加文件。

数据范围

对于20%的数据：$1 \leq n \leq 20$。

对于30%的数据：$1 \leq n \leq 30$。

对于60%的数据：$1 \leq n \leq 10000$。

对于额外20%的数据：保证$a_i = i$。

对于100%的数据：$1 \leq n \leq 2 \times 10^5, -10^5 \leq a_i \leq 10^5$。

思路分析

> 这是一道动态规划的题目，设$dp[i]$表示吃第i盘菜可以获取的最高美味程度，所以对于吃第j盘菜而言，如果j那么$dp[i]$是$dp[j]$的先前状态，就有$dp[j] = dp[i] + a[j]$。
>
> 可以分析出状态转移方程为

$$dp[i*j] = max(dp[i*j], dp[i] + a[i*j]) \qquad i*j <= n$$
$$dp[1] = max(a[1], 0)$$

答案取的结果为$max(dp[i]) \quad 1 \leq i \leq n$

> 可以用该状态转移方程写出程序

```cpp
#include <bits/stdc++.h>
using namespace std;
long long a[200001], dp[200001], ans, n;
int main() {
    cin >> n;
    for (int i = 1; i <= n; i++) cin >> a[i];
    dp[1] = max(a[1], 0ll);
    for (int i = 1; i <= n; i++) {
        for (int j = 2; i * j <= n; j++) {
            dp[i * j] = max(dp[i * j], dp[i] + a[i * j]);
        }
        ans = max(ans, dp[i]);
    }
    cout << ans;
    return 0;
}
```