信息学奥赛笔记04

考试题目汇总 | 贪心试题串讲

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。

```
```

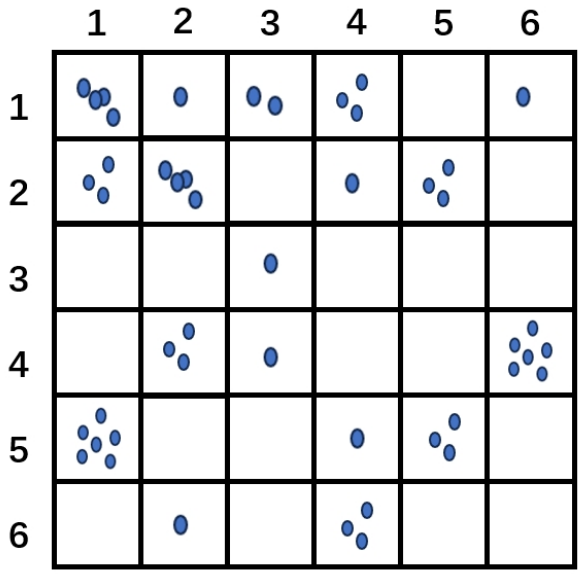
```cpp
#include <bits/stdc++.h>
using namespace std;
string s;
long long ans;
int main() {
    cin >> s;
    int n = s.size();
    for (int i = 0, j = 0; i < n; i++) {
        if (i && s[i] == s[i - 1]) j = i;  // 如果当前字符与相邻相等，则以当前这个作为起点
        ans += i - j + 1;   // 计算当前这个字符的贡献，并累加进答案中
    }
    cout << 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$1 \le n \le 10^5$，$1 \le m \le 10^9$，$1 \le a_i \le 10^9$。

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;
}
```

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;
}

```

删数问题

题目描述

一个集合有如下元素：1 是集合元素；若 P 是集合的元素，则 $2\times P+1$，$4\times P+5$ 也是集合的元素。

取出此集合中最小的 k 个元素，按从小到大的顺序组合成一个多位数，现要求从中删除 m 个数位上的数字，使得剩下的数字最大，编程输出删除前和删除后的多位数字。

注：不存在所有数被删除的情况。

输入格式

只有一行两个整数，分别代表 k 和 m。

输出格式

输出为两行两个整数，第一行为删除前的数字，第二行为删除后的数字。

样例 #1

样例输入 #1

```
5  4
```

样例输出 #1

```
137915
95
```

提示

数据规模与约定

- 对于 30% 的数据，保证 $1 \leq k, m \leq 300$。
- 对于 100% 的数据，保证 $1 \leq k, m \leq 3 \times 10^4$。

思路分析

> 这道题目需要拆分成两个问题来看，对于第一问，我们该求，如何获取集合中前k个数，这里不难想到可以用堆来进行求解以及优化

```cpp
int k, m;
priority_queue<int, vector<int>, greater<int> > pq;
pq.push(1);
cin >> k >> m;
while (k--) {
    int x = pq.top(); pq.pop(); // 获取堆最小的数
    s += to_string(x);  // 把最小的数转换成字符串加入答案s
    pq.push(2 * x + 1); // 把最小的数拓展2个数加入堆中
    pq.push(4 * x + 5);
}
cout << s << endl;  // 循环一共执行k遍，每次将最小的数加入答案s，s就储存了最小的k个数的字符串形式
```

> 这道题的难点其实在第二问上，该如何把一个串删除m位后剩余最大，其实思考上非常简单，只要保证高位最大就行，所以我们每次从前往后搜，只要在高位上出现一个逆序对，也就是小数在前，大数在后的情况，我们都可以删除小数，保留大数，将当前的操作循环m遍后，就是答案，其中删除字符串的某一位可以采用$erase(i, 1)$函数

```
while (m--) {
    for (int i = 0; i < s.size() - 1; i++) {
        if (s[i] >= s[i + 1]) continue;  // 如果是顺序对，则跳过
        s.erase(i, 1); // 逆序对的情况需要删除当前的最高位，程序直接跳出循环进入下一次删数。
        break;
    }
}
```

将两问的代码结合到一起的结果就是最终答案。

```
#include <bits/stdc++.h>
using namespace std;
int k, m;
string s;
priority_queue<int, vector<int>, greater<int> > pq;
int main() {
    cin >> k >> m;
    pq.push(1);
    while (k--) {
        int x = pq.top(); pq.pop();
        s += to_string(x);
        pq.push(2 * x + 1);
        pq.push(4 * x + 5);
    }
    cout << s << endl;  // 第一问输出
    while (m--) {
        for (int i = 0; i < s.size() - 1; i++) {
            if (s[i] >= s[i + 1]) continue;
            s.erase(i, 1);
            break;
        }
    }
    cout << s; // 第二问输出
    return 0;
}
```