信息学奥赛笔记14

新春收假考试试题解析

[U407234] 幸运单词 https://www.luogu.com.cn/problem/U407234

题目描述

小A每次做英语完形填空的时候都不知道该选什么。于是他想到一个方法可以猜一个答案。

这种方法的具体描述如下：假设 `maxn` 是单词中出现次数最多的字母的出现次数，`minn` 是单词中出现次数最少的字母的出现次数，如果 $\mathrm{maxn}-\mathrm{minn}$ 是一个偶数，那么小A就认为这个单词是幸运(Lucky)的，这样的单词很可能就是正确的答案。

输入格式

一个单词，其中只可能出现小写字母。

输出格式

共两行，第一行是一个字符串，假设输入的的单词是幸运的，那么输出 `Lucky`，否则输出 `No`；

第二行是一个整数，如果输入单词是 `Lucky` 的，输出 $\mathrm{maxn}-\mathrm{minn}$ 的值，否则输出 0。

样例 #1

样例输入 #1

```
error
```

样例输出 #1

```
Lucky
2
```

样例 #2

样例输入 #2

```
sally
```

样例输出 #2

```
No
0
```

提示

设n为输入字符串的长度。

对于100%的数据有，$1 \leq n \leq 10^5$，且保证数据均为小写字母。

【输入输出样例 1 解释】

单词 `error` 中出现最多的字母 `r` 出现了 3 次，出现次数最少的字母出现了 1 次，$3-1=2$，2 是偶数。

【输入输出样例 2 解释】

单词 `sally` 中出现最多的字母 `l` 出现了 2 次，出现次数最少的字母出现了 1 次，$2-1=1$，1 不是偶数。

> 本题在读完题后最直观的要求就是需要统计各个字母出现的个数，那么，我们会发现，我们需要统计的那个字母例如：`a`必须要和`a`出现的个数绑定在一起。两个变量的值需要一一对应关系的这种我们称之为叫**映射关系**。那么，如何在代码中处理映射关系呢？

我们需要使用到的方法是统计数组。

建立一个统计数组`cnt`，它的下标依次代表字母`a`到字母`z`出现的个数，通过维护这个数组来达到统计字母个数的目的。

举个例子，我们想要知道`a`这个字母出现了几次，那么我们可以建立一个数组，数组的值就代表每个字母出现的次数，但是，我们会发现，数组的下标不能是`a`，所以我不能通过`cnt['a']`的方式去访问次数，那可以怎么办呢？我们可以人为的为`a`编个号，比如说，就是数组的下标`0`，那我访问`0`的时候，就是在访问`a`，也就是在访问`a`出现的个数，那么同理，访问`cnt[25]`的值也就是访问的是`z`出现的次数。

那么在每次发现一个字母的时候，我们只需要做`cnt[s[i] - 'a']++`就可以了。

想要查找某个字母出现的次数也就直接访问`cnt[s[i] - 'a']`。

```cpp
#include <iostream>
#include <cmath>
#include <climits>
using namespace std;
int cnt[26];
int main() {
    string s;
    cin >> s;
    int n = s.size();
    int mx = 0, mn = INT_MAX; //调用<climits>库中的宏，INT_MAX就表示INT的最大值，也就是2147483647
    for (int i = 0; i < n; i++) {
        cnt[s[i] - 'a']++;
    }
    for (int i = 0; i < 26; i++) {
        if (cnt[i] == 0) continue; //如果该字母出现次数为0.则跳过
```

```
        mx = max(mx, cnt[i]);
        mn = min(mn, cnt[i]);
    }
    if ((mx - mn) % 2 == 0) {
        cout << "Lucky" << endl << mx - mn;
    } else cout << "No" << endl << "0";
    return 0;
}
```

[U406292] 中心下标 https://www.luogu.com.cn/problem/U406292

题目描述

给你一个长度为n整数数组`a`，请计算数组的 **中心下标** 。

数组 **中心下标** 是数组的一个下标，其左侧所有元素相加的和等于右侧所有元素相加的和。

如果中心下标位于数组最左端，那么左侧数之和视为 `0` ，因为在下标的左侧不存在元素。这一点对于中心下标位于数组最右端同样适用。

如果数组有多个中心下标，输出 **最靠近左边** 的那一个。如果数组不存在中心下标，输出 `-1` 。

输入格式

第一行一个正整数n，表示数组的长度

第二行n个整数，表示数组`a`

输出格式

一个整数，代表中心下标，如果没有则输出`-1`

样例 #1

样例输入 #1

```
6
1 7 3 6 5 6
```

样例输出 #1

```
3
```

提示

对于70%的数据，有$1 \leq n \leq 10^4$， $-10^4 \leq a[i] \leq 10^4$

对于100%的数据，有$1 \leq n \leq 10^5$， $-10^9 \leq a[i] \leq 10^9$

样例解释： 在下标为3时，左边和为$1+7+3=11$， 右边和为$5+6=11$。

90分做法

我们只需要按照题目的意思，模拟一遍，暴力搜索一个下标为中心下标，然后检查它的左边和是否等于右边和即可，开long long就可以拿90分。

```cpp
#include <iostream>
#include <vector>
#include <cmath>
#include <climits>
using namespace std;
int main() {
    int n;
    cin >> n;
    vector<long long> nums(n);
    for (int i = 0; i < n; i++) {
        cin >> nums[i];
    }
    for (int i = 0; i < n ; i++) {
        long long sum1 = 0, sum2 = 0; //sum1求左边和，sum2求右边和
        for (int j = 0; j < i; j++) {
            sum1 += nums[j];
        }
        for (int j = i + 1; j < n; j++) {
            sum2 += nums[j];
        }
        if (sum1 == sum2) { //左边和 == 右边和，i就是中心下标。
            cout << i << endl;
            return 0;
        }
    }
    cout << -1 << endl;
    return 0;
}

```

100分做法(该做法为前缀和 + 滑动窗口，不要求大家掌握)

那么，在暴搜的代码，咱们可以发现，对于同一段的和我们加了很多次重复的。

比如说当我们假设中心下标为`3`时，我们需要统计左边的和$a[0]+a[1]+a[2]$

但是我们在假设中心下标为`5`的时候，在统计左边的和时，依然需要计算上述式。

所以我们可以将整个数组分为三段，第一段`a`是中心下标左边的和，第二段`b`是中心下标本身，第三段`c`是中心下标右边的和。

所以我们可以发现，$a+b+c$永远等于整个数组的和，不管中心下标在哪。

所以我们在顺序遍历中心下标的时候，可以发现，设当前中心下标为`a[i]`，下一个中心下标为`a[i + 1]`

当前的左边的和为`sum`，整个数组的和为`total`，那么我们可以解得，右边的和为$total-sum-a[i]$

那如果左边的和等于右边的和就是判断$total - sum - a[i] == sum$则说明当前的`i`就是中心下标

那么从当前变动到下一个中心下标的时候，左边的和会增加刚刚我们处理的`a[i]`。

```cpp
#include <iostream>
#include <vector>
#include <cmath>
#include <climits>
using namespace std;
int main() {
    int n, ans = INT_MAX;
    cin >> n;
    vector<long long> nums(n);
    long long total = 0, sum = 0;
    for (int i = 0; i < n; i++) {
        cin >> nums[i];
        total += nums[i];
    }
    int c = 2;
    for (int i = 0; i < n; i++) {
        if (total - nums[i] == 2 * sum) {
            ans = min(ans, i);
        }
        sum += nums[i];
    }
    if (ans == INT_MAX) ans = -1;
    cout << ans << endl;
    return 0;
}
```

[U406860] 德州扑克II

题目背景

德州扑克，全名叫德克萨斯扑克，是一款风靡全球的桌面卡牌游戏。

注意：赌博有害，请远离赌博！！

题目描述

小G最近学习了德州扑克的玩法，作为一个初学者，他还没有完全记忆住德州扑克的比牌结果，他想让你帮他做一个训练程序，来帮助他快速的得出牌型。

德州扑克在最终的河牌圈，每名玩家要用手中的2张底牌去和桌面上的5张河牌组合出最大的**5张牌**的牌型，小G告诉你他组合出的那5张牌的点数和花色。并且他已经将手牌整理完毕，按照点数从小到大的顺序排好了。

德克萨斯扑克规则如下：

扑克牌一共有四种花色：梅花(C), 方块(D)，黑桃(S)，红桃(H)。

同种花色的点数共有13种，由[2, 3, 4, 5, 6, 7, 8, 9, 10, J, Q, K, A]组成，其中2最小，A最大。

手牌的5张牌可以组成**按照牌型大小从大到小**的：同花顺(Straight Flush)，四条(Four-of-a-Kind)，同花(Flush)，顺子(Straight)，三条(Three-of-a-Kind)，一对(One Pair)，高牌(High Card)

同花顺： 五张牌的花色相同，且点数从小到大依次增加1点，请输出"Straight Flush"

四条：四张相同点数的牌+一张单牌，请输出“Four-of-a-Kind”。

同花 ： 五张牌的花色相同，但是不可以组成顺子，请输出"Flush"。

顺子： 五张牌的花色不同，且点数从小到大依次增加1点，请输出"Straight"。

三条：三张点数相同的牌+两张点数不同的牌。请输出"Three-of-a-Kind"。

一对：含有一对相同点数的牌，剩下的牌点数都不相同，则输出"One Pair"。

高牌：不满足以上任意一种牌型，则输出"High Card"

输入格式

第一行包含一个字符串a，代表小G最终5张牌的点数，保证点数**从小到大**。

第二行包含一个长度为5的字符串b，代表小G手牌的花色

输出格式

一个字符串，表示小G的最大的5张手牌能组成的最大牌型。

样例 #1

样例输入 #1

```
910JQK
DDDDD
```

样例输出 #1

```
Straight Flush
```

样例 #2

样例输入 #2

```
35JQA
CSDHC
```

样例输出 #2

```
High Card
```

样例 #3

样例输入 #3

```
39999
CCDSH
```

样例输出 #3

```
Four-of-a-Kind
```

提示

其中保证a串仅由数字，$'J'$，$'Q'$，$'K'$，$'A'$组成。

保证b串仅由字符$'C'$，$'D'$，$'S'$，$'H'$组成。

> 和德州扑克一样，本题我们就不再探讨同花和顺子的解法了，和原版题相比，这次增加了四条三条一对的判断，那么，相当于我们需要记录一下最多出现的数牌它出现了一共几次，这和本次考试的第一题是有异曲同工之处的。我们需要一个统计数组，来记录最多出现的牌的数量，分别为四条三条一对设置一个标记，如果标记成立就说明这个牌型符合，从大到小判断并输出即可。

```cpp
#include <bits/stdc++.h>
using namespace std;
int main() {
    string s1, s2;
    cin >> s1 >> s2;
    int str = 0, flu = 0, fok = 0, tok = 0, op = 0, sum = 0;
    if (s2[0] == s2[1] && s2[0] == s2[2] && s2[0] == s2[3] && s2[0] == s2[4]) {
        flu = 1;
    }
    vector<int> nums, cnt(15);
    for (int i = 0; i < s1.size(); i++) {
        if (s1[i] >= '2' && s1[i] <= '9') nums.push_back(s1[i] - '0');
        if (s1[i] == '1') {
            nums.push_back(10);
            i++;
        }
        if (s1[i] == 'J') nums.push_back(11);
        if (s1[i] == 'Q') nums.push_back(12);
        if (s1[i] == 'K') nums.push_back(13);
        if (s1[i] == 'A') nums.push_back(14);
        int num = nums.back();
        ++cnt[num];    //统计当前这个数出现了几次
        if (cnt[num] == 4) {
            fok = 1;   //如果存在4次的牌，则四条标记为1
        }
        if (cnt[num] == 3) {
            tok = 1;
        }
        if (cnt[num] == 2) {
            op = 1;
        }
    }
    for (int i = 1; i < 5; i++) {
        sum += nums[i] - nums[i - 1] == 1;
    }
    if (sum == 4) str = 1;
    if (str == 1 && flu == 1) cout << "Straight Flush";  //从高到低逐个判断。
    else if (fok == 1) cout << "Four-of-a-Kind";
```

```
    else if (flu == 1) cout << "Flush";
    else if (str == 1) cout << "Straight";
    else if (tok == 1) cout << "Three-of-a-Kind";
    else if (op == 1) cout << "One Pair";
    else cout << "High Card";
    return 0;
}
```

[U406027] 版本号的绝对差 https://www.luogu.com.cn/problem/U406027

题目背景

在计算机软件中，我们习惯用版本号来标记不同时间下可以通过运行的代码。

题目描述

版本号的组成是一个字母V开头作为提示，由一个数字代表主版本号，和一个数字代表副版本号组成的，其中主副版本号之间用`.`分割。

例如：`V1234.5678`就是一个版本号，主版本号为`1234`，副版本号为`5678`。

输入数据中还有一个副版本号上限。副版本号等于上限时，该版本的下一个版本主版本号`+ 1`，副版本号清`0`。例如设副版本号的上限为`10`，那么`V1.10`的下一个版本就是`V2.0`， `V3.1`的上一个版本就是`V3.0`

现在给出一个软件的两个版本号，给出副版本号上限，请你求出这两个版本的绝对差是多少

注意：绝对差表示的是两个数的差值的绝对值。

输入格式

第一行包含一个字符串a，表示版本号1。

第二行包含一个字符串b，表示版本号2。

第三行k，表示副版本号的上限值。

保证输入的版本号一定是有效版本号。

输出格式

一行，表示版本号的绝对差。

样例 #1

样例输入 #1

```
V3.2
V3.0
5
```

样例输出 #1

```
2
```

样例 #2

样例输入 #2

```
V12.10
V15.7
11
```

样例输出 #2

```
33
```

提示

设$n = a.size(), m = b.size(), l = k.size()$

对于30%的数据，有$1 \leq n, m, l < 9$。

对于70%的数据，有$1 \leq n, m \leq 100, 1 \leq l \leq 5$。

对于100%的数据，有$1 \leq n, m, l \leq 5000$。

样例解释1： V3.0需要经过V3.1, V3.2，两个版本到V3.2

> 在做这道题的时候，大家需要先想明白一件事，13:20到18:40之间一共差了多少分钟，我们可以知道的是，一天一共有24小时，1440分钟，86400秒，我们想知道两个时间差了多少分钟，而且限定是一天的时间内，所以我们可以把任意一个分钟换算成**这是一天当中的第几分钟**，然后用分钟去相减，比如说刚刚的例子，13:20就是一天的第1*360 + 20 + 1 = 800*分钟，*18:40* 是*18*60 + 40 + 1 = 1120，1120 - 800 = 320分钟。这样就能得出结果。
>
> 那么我们来思考一下样例2的数据，每个版本号的副版本上限为11，说明在一个版本中有0-11一共12个版本。那么一小时60分钟，我们就用小时(h) * 60 + 分钟(m) + 1。一个版本有$k+1$个版本，那我们就应该换算成　主版本号 $*(k+1)+$副版本号 $+1$。
>
> 搞明白这个道理后我们来计算一下。
>
> V12.10就是第$12*12+10+1=155$。V15.7就是$15*12+7+1=188$，$188-155=33$。
>
> 那么我们只需要求出两个版本分别是第几个版本，然后做相减就做出来了。
>
> 那么这道题难在哪呢?
>
> 同学们会发现，版本的数位高达5000位，这就是在给大家提示，这道题应该使用高精度算法。
>
> 所以我们应该把运算的部分都修改成高精度即可。

```
#include <iostream>
#include <string>
#include <algorithm>
using namespace std;
```

```cpp
string add(string s1, string s2) {
    string ans;
    int l = s1.size() - 1, r = s2.size() - 1, carry = 0;
    while (l >= 0 || r >= 0) {
        int a = l >= 0 ? s1[l--] - '0' : 0;
        int b = r >= 0 ? s2[r--] - '0' : 0;
        carry += a + b;
        ans += carry % 10 + '0';
        carry /= 10;
    }
    if (carry) ans += "1";
    reverse(ans.begin(), ans.end());
    return ans;
}
string sub(string s1, string s2) {
    string ans;
    int l = s1.size() - 1, r = s2.size() - 1, carry = 0;
    if (l < r || l == r && s1 < s2) {
        swap(s1, s2);
        swap(l, r);
    }
    while (l >= 0 || r >= 0) {
        int a = l < 0 ? 0 : s1[l--] - '0';
        int b = r < 0 ? 0 : s2[r--] - '0';
        carry = a + 10 - b - carry;
        ans += carry % 10 + '0';
        carry = carry / 10 ^ 1;
    }
    while (ans.size() > 1 && ans.back() == '0') ans.pop_back();
    reverse(ans.begin(), ans.end());
    return ans;
}
string mul(string s1, string s2) {
    if (s1 == "0" || s2 == "0") return "0";
    int l1 = s1.size(), l2 = s2.size(), x;
    string ans(l1 + l2 - 1, '0');
    for (int i = l1 - 1; i >= 0; i--) {
        x = 0;
        for (int j = l2 - 1; j >= 0; j--) {
            int a = s1[i] - '0', b = s2[j] - '0';
            x += a * b + ans[i + j] - '0';
            ans[i + j] = x % 10 + '0';
            x /= 10;
        }
        if (i) ans[i - 1] += x;
    }
    if (x) ans = to_string(x) + ans;
    return ans;
}
int main() {
    string a1, b1, a2, b2, k;
    cin >> a1 >> b1 >> k; k = add(k, "1");
    int i = 1, j = 1;
    while (i < a1.size() && a1[i] != '.') {
        i++;
    }
    while (j < b1.size() && b1[j] != '.') {
        j++;
```

```
63
64      }
65      a2 = a1.substr(i + 1);
66      a1 = a1.substr(1, i - 1);
67      b2 = b1.substr(j + 1);
68      b1 = b1.substr(1, j - 1);
69      string x1 = add(mul(a1, k), a2);
70      string x2 = add(mul(b1, k), b2);
71      cout << sub(x1, x2);
72      return 0;
73  }
```