Problem A. 最小公倍数

难度	考点
2	gcd的使用，数据范围

题目分析

抓住a,b两数之积恰好等于最大公约数与最小公倍数之积这个思路进行求解，同时注意本题的数据范围，$1 \leq a, b \leq 1000000$,虽然它们本身在`int`范围内，但是其乘积确可以超过`int`范围，所以可以通过强制类型转换或者全部变量定义成`long long`，来保证数据不会因越界而出错。

示例代码

```
#include<stdio.h>
int main()
{

    int n;
    long long a,b;
    long long c,d;//c用来存放最大公约数，d用来存放最小公倍数
    long long e;//用来存放a,b的乘积
    long long temp;//临时存放数据
    int i=0;
    scanf("%d",&n);
    for(i=0;i<n;i++)
    {
        scanf("%lld%lld",&a,&b);
        e=a*b;
        while(b!=0)//通过辗转相除法求最大公约数
        {
            temp=a%b;
            a=b;
            b=temp;
        }
        c=a;
        d=e/c;//a,b两数之积恰好等于最大公约数与最小公倍数之积
        printf("%lld\n",d);
    }
    return 0;
}
```

Problem B. Liella!

难度	考点
1	数组、循环

题目分析

见代码，按题目要求统计成绩即可。

示例代码

```
#include <stdio.h>
int min = 105, max = -1, cnt, sum, mode, maxmode = -1, x, maxi, mini;
int score[105], i;
int main()
{
    while (scanf("%d", &x) != EOF)
    {
        if (x == -1)
        {
            break;
        }
        else if (x < -1 || x > 100)
        {
            continue;
        }
        //每输入一个数
        cnt++;//同学数++
        score[x]++;//哈希表中，这个分数出现的次数++
        sum += x;//总分+=x
        //维护最值
        if (x >= max)
        {
            max = x;
            maxi = cnt;
        }
        if (x <= min)
        {
            min = x;
            mini = cnt;
        }
    }
    //遍历哈希表，求众数
    for (i = 0; i <= 100; i++)
    {
        if (score[i] >= maxmode)
        {
            maxmode = score[i];
            mode = i;
        }
    }
    printf("count: %d\n", cnt);
    printf("ave: %.4f\n", 1.0 * sum / cnt);
    printf("max: %d: %d\n", maxi, max);
    printf("min: %d: %d\n", mini, min);
    printf("mode: %d\n", mode);
    return 0;
}
```

Problem C. 小鲨的学习大法

难度	考点
2	循环、基础运算

题目分析

题目要求$K \times M + L = N$，我们直接暴力枚举K, M使得$K \times M < N$即可。

示例代码

```
#include <stdio.h>
int main(){
    int k,m,n,sum=0;
    scanf("%d",&n);
    for(k=1;k<n;k++)
        for(m=1;m*k<n;m++)sum++;
    printf("%d",sum);
    return 0;
}
```

Problem D. 水水的浮点数二进制

题目分析

整数部分：就是正常的进制转换，可以参考课件和百度。
小数部分：如下，模拟题目描述的小数转二进制的过程，不断将小数乘 2 并取整数部分。

示例代码

```
#include <stdio.h>
double x, temp;
int i, a, a1, b[33], n;
int main()
{
    scanf("%lf", &x);
    a = x;
    a1 = a;
    while (a != 0)
    {
        b[n] = a % 2;
        a = a / 2;
        n++;
    }
    if (a1 == 0)
    {
```

```
        printf("0");
    }
    else
    {
        for (i = n - 1; i >= 0; i--)
        {

            printf("%d", b[i]);
        }
    }
    printf(".");
    //以上是整数部分，下面是小数
    x = x - a1;//x是零开头的小数
    for (i = 0; i < 10; i++)
    {
        temp = x * 2;
        if (temp >= 1)
        {
            printf("1");
            x = temp - 1;
        }
        else
        {
            printf("0");
            x = temp;
        }
    }
    return 0;
}
```

Problem E. 超可爱的XIAO7和RGB字符串

难度	考点
3	暴力枚举

题目分析

因为字符串的形式只能形如`rrrr...ggggg...bbbb`，且长度为 n 已固定，因此只要`r`和`g`的数量确定，整个字符串就确定了。因此可以得到如下做法：

枚举`r`和`g`的数量的所有情况，即最终得到的`可爱的字符串`的所有情况，然后与初始字符串进行比对，计算需要修改多少个字符，最终答案取修改次数最小的情况即可。

示例程序

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<math.h>
#include<stdbool.h>
```

```
#define N 105
char str[N];
int main(){
    int n=0,ans=N;
    while((str[++n]=getchar())!='!');
    n--;//长度减一,删去最后的!
    for(int r=0;r<=n;r++)//r的数量
        for(int g=0;r+g<=n;g++){//g的数量
            int cnt=0;//记录当前情况需要修改的字符数
            for(int i=1;i<=r;i++)//前1-r个字符最终状态应该是r
                cnt+=(str[i]!='r');//如果不是r需要修改,修改次数+1
            for(int i=r+1;i<=r+g;i++)//第r+1到g+1个字符是g
                cnt+=(str[i]!='g');//如果不是g需要修改,修改次数+1
            for(int i=r+g+1;i<=n;i++)//第r+g+1到n个字符是b
                cnt+=(str[i]!='b');//如果不是b需要修改,修改次数+1
            ans=cnt<ans?cnt:ans;//与当前ans比较取较小值
        }
    printf("%d\n",ans);
    return 0;
}
```

Problem F. 暴的不是很力

难度	考点
3	暴力枚举

题目分析

如果本题用三重循环枚举所有的三个三位数，总循环次数将达到 10^8 到 10^9 之间，会 TLE，因此要利用题目条件减少枚举次数。

题目说明中给出 A, B, C 之间互素，因此**任意一组解均可表示为 $A*i, B*i, C*i$ 的形式。**所以只需要枚举 i，并且判断每组数据是否正好包含了 $1 \backsim 9$。

对于判断是否包含了 $1 \backsim 9$，可以将每组三个数逐个拆开，用数组 $mark[i]=1$ 表示数字 i 出现过。若存在 $i(1 \leq i \leq 9)$ 使得 $mark[i]$不为 1，则不满足条件，跳过此轮循环；否则输出该组数据。

用变量 $find$ 表示是否发现可行解，如果没有可行解则在程序结尾处输出 `NOT FOUND`。

示例代码

```
#include <stdio.h>

int main()
{
    int A, B, C;
    scanf("%d %d %d", &A, &B, &C);
    int begin = 123 / A, end = 987 / C; //求出i的上下界，减少枚举次数
    int mark[10], num[3], find = 0;
    for (int i = begin; i <= end; i ++)
    {
```

```
        num[0] = A * i, num[1] = B * i, num[2] = C * i; //用num数组存储每组A*i, B*i, C*i三个
数
        for (int i = 1; i <= 9; i ++) //每轮循环开始时将mark数组清空
            mark[i] = 0;
        for (int j = 0; j < 3; j ++)
            mark[num[j] % 10] = mark[num[j] / 10 % 10] = //三位数的拆分
            mark[num[j] / 100] = 1;
        int flag = 0;
        for (int j = 1; j <= 9; j ++)
            if (mark[j] == 0)
                flag = 1;
        if (flag) continue;
        for (int j = 0; j < 3; j ++)
            printf("%d ", num[j]);
        puts("");
        find = 1;
    }
    if (!find) printf("NOT FOUND");
    return 0;
}
```

Problem G. 士谔2173&&传源2171_PLUS

难度	考点
4	数组、枚举

题目分析

按题目要求求出区间中的每个数的 n_code1 和 n_code2 ，判断其是否是**士谔传源数**，并与之前C系列做过的简化版一样，用一个`vis`数组标记其是否出现过，并输出结果即可。

既然现在已经学过函数了，n_code1 和 n_code2 的计算是不是写成一个函数会更方便呢~

示例程序

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<math.h>
#include<stdbool.h>
#define N 20005
bool vis[N];
int code1(int x){//计算n_code1
    int cnt=0;
    for(;x;x>>=1) cnt+=(x&1);
    return cnt;
}
int code2(int x){//计算n_code2
    for(int y=x,num=0;x>9;x=num)
        while(y) num+=y%10,y/=10;
```

```
    return x;
}
bool check(int x){//检查数字x是否为2,1,7
    return x==2 || x==1 || x==7;
}
int main(){
    int T,L,R;
    scanf("%d",&T);
    while(T--){
        scanf("%d%d",&L,&R);
        bool first=1;
        for(int i=L;i<=R;i++)if(!vis[i]){//跳过已经输出过的数
            int cnt=0,x=i;
            while(x)//计算数字i出现2,1,7的位数数量
                cnt+=check(x%10),x/=10;
            if(check(code1(cnt)) || check(code2(cnt))){
                if(first) first=0;
                else putchar(32);//空格的控制
                printf("%d",i),vis[i]=1;
            }
        }
        putchar(10);
    }
    return 0;
}
```

Problem H. 科学计数法

难度	考点
2	字符串的应用

题目分析

首先，题目明确了输入为正数，则不需要考虑负号。

其次，题目保证当小数点出现时，其左右两边均有数字，但未说明数字是否有多余的`0`，故需要特别判断。

最后，当没有小数部分时不输出小数点。次数为`0`时不输出`E`。

结合上述条件，提供一个比较简明的思路供参考：

1，把数字“拉通”看，去掉开头和末尾多余的`0`。具体而言，是要分别求出从左到右和从右到左第一个非`0`的数字。这样可以减少讨论情形。

2，进行上述操作后，就相当于在该数字中多了一个小数点而已，此时找到它的位置即可确定幂次。

3，特别考虑结果为`0`、次数为`0`的情况。

而在处理数字本身时需要注意的特别情况有：

1，整数部分左端有多余的`0`。

2，小数部分右端有多余的`0`。

3，整数部分仅有多个`0`。

4，小数部分仅有多个`0`。

示例程序

```
#include<stdio.h>
#include<string.h>
char s[25];
int main(){
    int i, len, p_dot, first_num, last_num, Pow;

    gets(s);
    len = strlen(s);
    for(i = 0; i < len; i++){
        if(s[i] == '.') break;
    }
    p_dot = i;  //求出小数点的位置，若没有则为len。

    for(i = 0; i < len; i++){
        if(s[i] != '0' && s[i] != '.') break;
    }
    first_num = i; //求出从左到右第一个非0数的位置，若没有则为len。

    for(i = len - 1; i >= 0; i--){
        if(s[i] != '0' && s[i] != '.') break;
    }
    last_num = i; //求出从右到左第一个非0数的位置，若没有则为len。

    if(last_num <= p_dot && first_num >= p_dot){ //此时说明全是0，直接输出0。
        putchar('0');
        return 0;
    }

    if(first_num < p_dot) Pow = p_dot - first_num - 1;
    else Pow = p_dot - first_num; //求出次数。

    putchar(s[first_num]); //后面为输出结果。
    if(last_num > first_num){
        putchar('.');
        for(i = first_num + 1; i <= last_num; i++)
            if(s[i] != '.') putchar(s[i]);
    }

    if(Pow != 0) printf("E%d", Pow);

    return 0;
}
```

Problem I. 古墓丽影——Euler's Legacy

难度	考点
3	简单循环，数据类型

题目分析

做法显然是用 HINT 里给的欧拉函数进行计算，注意数据范围可达 10^{15} 量级，所以要使用 `long long` 的数据类型。

关于欧拉函数的计算，朴素的想法是寻找 n 的所有素因子，然后再累乘，问题在于这个素因子判断的过程，比如遍历 $2 \to \sqrt{n}$，是每个数都要检查一下是素数吗，还是另有他法？（事实上每个数都判断一遍会超时）

这里就要采取算术基本定理的思想，众所周知正整数 n 有唯一的分解表示：$n = p_1^{k_1} \dots p_m^{k_m}$，如果每次判断为素数后我们继续把 n 包含的剩下的相同素因子给除掉，这样的结果就是**每次判断 `n % i == 0` 一定能保证这样的 `i` 只会是素数。**

如果加粗部分理解还不直观的话，可以公式说明一下：

> $n = p_1^{k_1} \dots p_m^{k_m}$；
>
> 整个循环过程第一次遇到的 i 满足 $n \equiv 0 \pmod{i}$ 的必然有 $i = p_1$；
>
> 除掉所有 p_1 因子后，此时 $n' = p_2^{k_2} \dots p_m^{k_m}$；
>
> 下一次遇到的第一个 i 满足 $n' \equiv 0 \pmod{i}$ 时，必然是 $i = p_2$；
>
> 后面依此类推……

需要注意的是当 n 是一个素数时，循环完成后需要单独判断，当然事实上有 $\varphi(p) = p - 1$，p是一个素数。

示例程序

```
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
typedef long long LL;
LL euler_phi(LL n)
{
    LL m = sqrt(n + 0.5);
    LL ans = n;
    for (LL i = 2; i <= m; ++i)
    {
        if (n % i == 0)
        {
            ans = ans / i * (i - 1);
            while (n % i == 0)
                n /= i;
        }
    }
    if (n > 1)
        ans = ans / n * (n - 1); // if n is a prime
    return ans;
}
int main()
{
    LL n;
```

```
    scanf("%lld", &n);
    printf("%lld\n", euler_phi(n));
    return 0;
}
```

复杂度分析

标程

按照标程的做法，计算其欧拉函数值，外层循环从 $2 \to \lfloor \sqrt{n} \rfloor$，内层实际上是做的一个将 n 分解为按算术基本定理的唯一形式 $n = p_1^{k_1} \dots p_m^{k_m}$ 的过程，分解过程复杂度为 $O(\sqrt{n})$，注意这两个过程不是外层套内层，而是同时进行的，所以复杂度是两者相加最终为：$O(\sqrt{n})$

常见 TLE 问题分析

- 暴力遍历每个比 n 小的数，求 $\gcd(i, n)$，采用辗转相除的写法复杂度为 $O(\log n)$，整体为 $O(n \log n)$。
- 外层循环为 $2 \to \lfloor \sqrt{n} \rfloor$，但是内层没有按照算术基本定理的思想分解 n，而是每次都判断一次是否为素数，素数判断 $O(\sqrt{n})$ ，此时外层套内层，总复杂度为 $O(n)$。
- 当然所谓的骗分点的精髓在于p是素数时，有$\varphi(p) = p - 1$，也就是素数的大数据点只需要用$o(\sqrt{n})$的复杂度判断是否为素数就可以直接输出结果。

Problem J. 罪魁祸首 题解

难度	知识点
4/10	数组、思维

这个题乍一看其实很简单，有三种一眼就能看出来的思路:

1. 顺序扫描一遍输入的序列，用一个数组cnt记录编号为i的魔兽出现的次数。之后从cnt_i到$cnt_{\max\{a_i\}}$中找到最大值，并判断其是否大于$\frac{n}{2}$即可。

   这种做法可以得到70分，因为对于第$8, 9, 10$组数据，a_i会非常的大，以至于开不了这么大的数组。（MLE或者REG）

2. 按顺序取数组中的数，并判断这个数出现的频率是否超过50%，如果是就输出，如果不是就另取一个数，直到发现出现频率超过50%的数，或者取遍数组中的所有数为止。

   平均来说，这种做法的时间复杂度是$O(n)$，因为任意取一个数，如果所求的数存在，那么一次选到的概率超过$1/2$，选择次数的期望值（平均值）不超过2。但是最坏情况下可以达到$O(n^2)$（例如根本不存在所求的数，或者一直恰好怎么也选不到所求的数），因此在面对某些数据时会TLE。

3. 直接将数组排序，那么如果存在题目所求的数字，一定是排完序的数组的中位数。

   （其实咱们还没学排序，而且这也不是正解，所以没有兴趣的同学可以直接略过这一部分了）

   如果使用C语言自带的快速排序函数，其复杂度为$O(n \log n)$，在n取最大值时，运算量可以接近10^8，对于评测机来说，一秒钟运算完可能会有些吃力，有可能会TLE。

那么**正解**是什么呢?

我们需要注意到这样一个事实：如果一个序列中存在出现频率超过50%的数，那么在这个序列中删除任意两个不相同的数并不会改变这个出现频率超过50%的数。

事实上：假设序列长度为n，这个数出现的频数是k，当$k > \frac{n}{2}$时，有：

$$\text{出现频率} = \frac{k}{n} < \frac{k-1}{n-2} < \frac{k}{n-2}$$

因此，我们只需要顺序扫描一遍数组，遇到一对儿不相等的数就删掉，那么剩下的就是要求的数。

说得轻松，怎么实现“遇到一对不相等的就删掉”呢？我们可以另开一个新数组，并按顺序把原先数组中的数往新数组里放，但是一旦即将放进去的数和新数组中最新的数不一样，就把两个数同时删去。

现举例说明如下：

原数组为：

$$4, 4, 1, 2, 3, 4, 4$$

扫描下标i	"新数组"	说明
1	4	
2	4，4	
3	4	1与新数组中最新数4（a_2）不同，将两个都删去
4	空	2与新数组中最新数4（a_1）不同，将两个都删去
5	3	
6	空	4与新数组中最新数3（a_5）不同，将两个都删去
7	4	输出答案：4

在实际编码时，我们并不需要真的去开一个“新数组“，这是因为新数组中的数要不没有（空），要不是同一个数的重复。因此我们只需要使用两个数字变量，一个用来记录新数组的数值，另一个用来记录这个新数组的长度就行了。

这种方法只需要顺序扫描2遍数组，复杂度为$O(n)$.

示例程序

```
#include <stdio.h>
#include <string.h>
#include <math.h>
#include <ctype.h>
#include <stdlib.h>
#define ll long long
#define maxn 5000000+10
const double eps=1e-11;

int n,a[maxn];

int main(){
    scanf("%d",&n);
    for(int i=1;i<=n;++i)
```

```
        scanf("%d",&a[i]);
    int temp=a[1],cnt=1;//temp记录新数组数值，cnt记录新数组长度
    for(int i=1;i<=n;++i){
        if(cnt==0)
            temp=a[i];
        if(a[i]==temp)
            ++cnt;
        else
            --cnt;
    }
    int Cnt=0;
    for(int i=1;i<=n;++i)//检验是否超过50%
        if(a[i]==temp)
            ++Cnt;
    if(Cnt<=n/2)
        printf("Not Found\n");
    else
        printf("%d\n",temp);
    return 0;
}
```