问题A: OH哥的倍数问题（Easy Version）

> 本系列问题公倍数用lcm表示

可以先计算一下n以内a的倍数的和
首项 a，尾项 n / a * a ，个数 n / a，公式：(a + n / a * a) * (n / a) / 2
同理b的是 (b + n / b * b) * (n / b) / 2
发现重复计算了a和b的公倍数，那么就加回来，(lcm(a, b) + n / lcm(a, b) * lcm(a, b)) * (n / lcm(a, b)) / 2

```
#include <stdio.h>
#define ll long long

ll n;

inline ll get ( ll x ) {
        return ( x + n / x * x ) * (n / x) / 2;
}
inline ll gcd ( ll a, ll b ) {
    return b ? gcd ( b, a % b ) : a;
}
inline ll lcm ( ll a, ll b ) {
    return a * b / gcd(a, b);
}

int main () {
        int cass;
        for ( scanf("%d", &cass); cass; cass -- ) {
                scanf("%lld", &n);
                ll a, b; scanf("%lld%lld", &a, &b);
                printf("%lld\n", get(a) + get(b) - get(lcm(a, b)));
        }
}
```

问题B: OH哥的倍数问题（Hard Version）

是上一个问题的扩展版本
a的倍数b的倍数c的倍数d的倍数e的倍数加上后，重复计算了lcm(两个数)的倍数，要删去
同时删掉后多删去了lcm(三个数)的倍数，要加上
同时加上后多加上了lcm(四个数)的倍数，要减去
同时减去后多减去了lcm(五个数)的倍数，要加上

```cpp
#include <stdio.h>
#define ll long long

ll n;

inline ll get ( ll x ) {
        return ( x + n / x * x ) * (n / x) / 2;
}
inline ll gcd ( ll a, ll b ) {
    return b ? gcd ( b, a % b ) : a;
}
inline ll lcm ( ll a, ll b, ll c, ll d, ll e ) {
    ll x = a * b / gcd(a, b);
    x = x * c / gcd(x, c);
    x = x * d / gcd(x, d);
    x = x * e / gcd(x, e);
    return x;
}

int main () {
        int cass;
        for ( scanf("%d", &cass); cass; cass -- ) {
                scanf("%lld", &n);
                ll a, b, c, d, e;
                scanf("%lld%lld%lld%lld%lld", &a, &b, &c, &d, &e);
                ll res = 0;
                res = res + get(a) + get(b) + get(c) + get(d) + get(e);
                res = res - (get(lcm(a, b, 1, 1, 1)) + get(lcm(a, c, 1, 1, 1)) + get(lcm(a, d, 1, 1, 1)) + get(lcm(a, e, 1, 1, 1)) +
                             get(lcm(b, c, 1, 1, 1)) + get(lcm(b, d, 1, 1, 1)) + get(lcm(b, e, 1, 1, 1)) +
                             get(lcm(c, d, 1, 1, 1)) + get(lcm(c, e, 1, 1, 1)) +
                             get(lcm(d, e, 1, 1, 1)));
                res = res + get(lcm(a, b, c, 1, 1)) + get(lcm(a, b, d, 1, 1)) + get(lcm(a, b, e, 1, 1)) +
                             get(lcm(a, c, d, 1, 1)) + get(lcm(a, c, e, 1, 1)) +
                             get(lcm(a, d, e, 1, 1)) +
                             get(lcm(b, c, d, 1, 1)) + get(lcm(b, c, e, 1, 1)) +
                             get(lcm(b, d, e, 1, 1)) +
                             get(lcm(c, d, e, 1, 1));
                res = res - get(lcm(a, b, c, d, 1)) - get(lcm(a, b, c, e, 1)) - get(lcm(a, b, d, e, 1)) - get(lcm(b, c, d, e, 1)) - get(lcm(a, c, d, e, 1));
                res = res + get(lcm(a, b, c, d, e));
                printf("%lld\n", res);

        }
}
```

the,hello的二进制枚举解法

```cpp
#include <bits/stdc++.h>
using namespace std;
bool pd[1086];
long long sum(long long  a,int n){
    long long len=n/a;
    return (a*len*(len+1))>>1;
}
int gcd(int a,int b){
    if(b==0)return a;
    return gcd(b,a%b);
}
long long lcm(int a,int b){
    int gcd_=gcd(a,b);
    return 1LL*a*b/gcd_;
}
int main()
{
   int t;
   scanf("%d",&t);
   while (t--)
   {
        int n,a[6],len=0;
        long long sum_=0;
        memset(pd,0,sizeof(pd));
        scanf("%d",&n);
        for(int i=1,j;i<=5;i++){
            scanf("%d",&j);
            if(!pd[j]){
                pd[j]=1;
                a[++len]=j;
            }
        }
        if(pd[1]){
            printf("%lld\n",sum(1,n));
            continue;
        }
        int tot=(1<<len)-1;
        for(int i=1;i<=tot;i++){
            long long num=1;
            int statu=0;
            for(int j=1;j<=len;j++)
                if((i>>(j-1))&1){
                    statu+=1;
                    num=lcm(a[j],num);
                }
            if(statu%2!=0)statu=1;
            else statu=-1;
            sum_+=statu*sum(num,n);
        }
        printf("%lld\n",sum_);
   }

}
```

问题C： 浅谈—如何更简单地设计程序

通过平方差公式展开分母，再和分子最后一项相消

$$\frac{a^{2^x}-b^{2^x}}{(a^{2^0}+b^{2^0})(a^{2^1}+b^{2^1})(a^{2^2}+b^{2^2})(a^{2^3}+b^{2^3})...(a^{2^{x-1}}+b^{2^{x-1}})}=\frac{(a^{2^{x-1}}+b^{2^{x-1}})(a^{2^{x-1}}-b^{2^{x-1}})}{(a^{2^0}+b^{2^0})(a^{2^1}+b^{2^1})(a^{2^2}+b^{2^2})(a^{2^3}+b^{2^3})...(a^{2^{x-1}}+b^{2^{x-1}})}$$

$$=\frac{a^{2^{x-1}}-b^{2^{x-1}}}{(a^{2^0}+b^{2^0})(a^{2^1}+b^{2^1})(a^{2^2}+b^{2^2})(a^{2^3}+b^{2^3})...(a^{2^{x-2}}+b^{2^{x-2}})}=\frac{(a^{2^{x-2}}+b^{2^{x-2}})(a^{2^{x-2}}-b^{2^{x-2}})}{(a^{2^0}+b^{2^0})(a^{2^1}+b^{2^1})(a^{2^2}+b^{2^2})(a^{2^3}+b^{2^3})...(a^{2^{x-2}}+b^{2^{x-2}})}$$

$$=\frac{a^{2^{x-2}}-b^{2^{x-2}}}{(a^{2^0}+b^{2^0})(a^{2^1}+b^{2^1})(a^{2^2}+b^{2^2})(a^{2^3}+b^{2^3})...(a^{2^{x-3}}+b^{2^{x-3}})}=....$$

....

可得，分子每一层指数x可以和分母的指数x-1相消
所以最后可以只剩下分子的$a^{2^0} - b^{2^0} = a - b$
那么$a - b$可能出现负数，就 `(a % mod - b % mod + mod ) % mod` 即可

```
#include <stdio.h>

const int mod = 1e9 + 7;

int main () {
        int t; scanf("%d", &t); while ( t -- ) {
                long long x, a, b; scanf("%lld%lld%lld", &x, &a, &b);
                a %= mod, b %= mod;
                printf("%lld\n", (a + mod - b) % mod);
        }
}
```

问题D： 旭旭又来了

签到题懂的都懂

```
#include <stdio.h>

int main()
{
    for (int i = 0; i < 10; i++)
        printf("xuxu loves Honor of Kings %%%%%%\n");
    return 0;
}
```

问题E： 旭旭双来了

观察规律，可以分成上下两部分打印

```
#include <stdio.h>

int main()
{
    int n;
    scanf("%d", &n);
    int ch1 = 0, ch2 = (n - 1) % 26; //ch2即图形右上角第一个字母
    n = n / 2 - 1;
    for (int i = n; i >= 0; i--) //上半部分
    {
        for (int j = i; j < n; j++) printf(" ");
        printf("%c", ch1 + 'A');
        ch1 = (ch1 + 1) % 26;
        for (int j = 0; j < i * 2; j++) printf(" ");
        printf("%c\n", ch2 + 'A');
        ch2 = (ch2 + 25) % 26;
    }
    for (int i = 0; i <= n; i++) //下半部分
    {
        for (int j = i; j < n; j++) printf(" ");
        printf("%c", ch2 + 'A');
        ch2 = (ch2 + 25) % 26;
        for (int j = 0; j < i * 2; j++) printf(" ");
        printf("%c\n", ch1 + 'A');
        ch1 = (ch1 + 1) % 26;
    }
    return 0;
}
```

问题F： 旭旭叒来了

签到题，注意多实例，输入有负数，NO是两个字母都大写

```
#include <stdio.h>
#include <math.h>
int check(int x)
{
    if (x < 2) return 0;
    for (int i = 2; i <= sqrt(x); i++)
        if (x % i == 0) return 0;
    return 1;
}
int main()
{
    int x;
    while (~scanf("%d", &x))
        printf("%s\n", check(x) ? "Yes" : "NO");
    return 0;
}
```

问题G： 旭旭叕来了

最基础的计算日期问题，给大家提供模板

```
#include <stdio.h>

int a[2][13] = {{0, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31}, {0, 31, 29, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31}};
//可以直接表示某月有多少天

int leap(int n) //判断闰年
{
    if (n % 400 == 0 || n % 100 && n % 4 == 0) return 1;
    return 0;
}

int main()
{
    int y1, y2, m1, m2, d1, d2;
    while (~scanf("%d%d%d%d%d%d", &y1, &m1, &d1, &y2, &m2, &d2))
    {
        int ans = 0;
        for (int i = y1; i < y2; i++) ans += leap(i) ? 366 : 365; //先计算差了多少年
        for (int i = 1; i <= m1; i++) //计算day1距1月1日的天数
        {
            if (i == m1) ans -= d1;
            else ans -= a[leap(y1)][i];
        }
        for (int i = 1; i <= m2; i++) //计算day2距1月1日的天数
        {
            if (i == m2) ans += d2;
            else ans += a[leap(y2)][i];
        }
        printf("%d\n", ans);
    }
    return 0;
}
```

问题H： 旭旭不走了

在两个0之间计算即可。

```c
#include <stdio.h>

char s[1000010];
int k;

int main()
{
    s[0] = '0'; //将第一个数设为0，方便后面计算
    while (~scanf("%s%d", s + 1, &k)) //字符串从s+1开始读
    {
        long long ans = 0; //注意k最大为19，计算结果用long long保存
        for (int i = 0; s[i]; i++)
        {
            if (s[i] == '0') //找到一个0开始的区间
            {
                int flag = 1, pos; //pos记录当前位置，flag记录是否找到长度大于等于k的区间
                long long fact = 1; //记录乘积
                for (pos = i + 1; pos <= i + k; pos++)
                {
                    if (s[pos] == '0' || s[pos] == '\0') //区间长度不足k，直接退出
                    {
                        flag = 0;
                        break;
                    }
                    fact *= (s[pos] - '0'); //计算乘积
                }
                if (flag) //此时已经得到k个连续数字的乘积
                {
                    if (ans < fact) ans = fact; //更新最大值
                    for (; s[pos] && s[pos] != '0'; pos++) //只要不是0继续遍历
                    {
                        fact = fact / (s[pos - k] - '0') * (s[pos] - '0'); //计算乘积
                        if (ans < fact) ans = fact; //更新最大值
                    }
                }
                i = pos - 1; //更新当前位置
            }
        }
        printf("%lld\n", ans);
    }
    return 0;
}
```

问题I 旭旭被赶走了

两个大数字相乘，都可以拆分成多个质数相乘，而质数相乘结果尾数为0的，只可能是2*5。如果想到了这一点，那么就可以进一步想到：两个数相乘尾数0的个数其实就是依赖于2和5因子的个数。又因为每两个连续数字就会有一个因子2，个数非常充足，所以此时只需要关心5因子的个数就行了。

标程是用二分，但是某巨佬用奇妙的方法解决了这个问题。先放标程，可以在学习二分算法之后再来理解标程

```cpp
#include <stdio.h>

typedef long long ll;

ll num(ll x) //计算x的阶乘的末尾0的个数
{
    ll t = 5, sum = 0;
    while (t <= x)
    {
        sum += x / t;
        t *= 5;
    }
    return sum;
}

int main()
{
    int m, n, i, j;
    ll mid, l = 0, r = 1e18;
    scanf("%d", &m);
    while (l < r)
    {
        mid = (l + r + 1) / 2;
        if (num(mid) >= m) r = mid - 1;
        else l = mid;
    }
    printf("%lld", l + 1);

    return 0;
}
```

某巨佬的奇妙方法
每次加1000，计算时间大幅缩小

```cpp
#include <stdio.h>

typedef long long LL;

LL pan0(LL num)
{
        LL cnt = 0;
        while (num > 0)
        {
                cnt += num / 5;
                num /= 5;
        }
        return cnt;
}

int main()
{
        LL m;
        scanf("%lld", &m);
        for (LL i = 1;; i++)
        {
                if (pan0(i) < m)
                {
                        if (pan0(i + 1000) < m) i += 1000;
                        else if (pan0(i + 900) < m) i += 900;
                        else if (pan0(i + 800) < m) i += 800;
                        else if (pan0(i + 700) < m) i += 700;
                        else if (pan0(i + 600) < m) i += 600;
                        else if (pan0(i + 500) < m) i += 500;
                        else if (pan0(i + 400) < m) i += 400;
                        else if (pan0(i + 300) < m) i += 300;
                        else if (pan0(i + 200) < m) i += 200;
                        else if (pan0(i + 100) < m) i += 100;
                }
                if (pan0(i) >= m)
                {
                        printf("%lld", i);
                        return 0;
                }
        }
        return 0;
}
```