ACM-ICPC 培训资料汇编

（6）

数论、组合数学分册

（版本号 **1.0.0**）

哈尔滨理工大学 ACM-ICPC 集训队

2012 年 12 月

序

2012 年 5 月，哈尔滨理工大学承办了 ACM-ICPC 黑龙江省第七届大学生程序设计竞赛。做为本次竞赛的主要组织者，我还是很在意本校学生是否能在此次竞赛中取得较好成绩，毕竟这也是学校的脸面。因此，当 2011 年 10 月确定学校承办本届竞赛后，我就给齐达拉图同学很大压力，希望他能认真训练参赛学生，严格要求受训队员。当然，齐达拉图同学半年多的工作还是很有成效，不仅带着黄李龙、姜喜鹏、程宪庆、卢俊达等队员开发了我校的 OJ 主站和竞赛现场版 OJ，还集体带出了几个比较像样的新队员，使得今年省赛我校取得了很好的成绩（当然，也承蒙哈工大和哈工程关照，没有派出全部大牛来参赛）。

在 2011 年 9 月之前，我对 ACM-ICPC 关心甚少。但是，我注意到我校队员学习、训练没有统一的资料，也没有按照竞赛所需知识体系全面系统培训新队员。2011-2012 年度的学生教练们做了一个较详细的培训计划，每周都会给 2011 级新队员上课，也会对老队员进行训练，辛辛苦苦忙活了一年——但是这些知识是根据他们个人所掌握情况来给新生讲解的，新生也是杂七杂八看些资料和做题。在培训的规范性上欠缺很多，当然这个责任不在学生教练。2011 年 9 月，我曾给老队员提出编写培训资料这个任务，一是老队员人数少，有的还要去百度等企业实习；二是老队员要开发、改造 OJ；三是培训新队员也很耗费精力，因此这项工作虽很重要，但却不是那时最迫切的事情，只好被搁置下来。

2012 年 8 月底，2012 级新生满怀梦想和憧憬来到学校，部分同学也被 ACM-ICPC 深深吸引。面对这个新群体的培训，如何提高效率和质量这个老问题又浮现出来。市面现在已经有了各种各样的 ACM-ICPC 培训教材，主要算法和解题思路都有了广泛深入的分析和讨论。同时，互联网博客、BBS 等中也隐藏着诸多大牛对某些算法的精彩论述和参赛感悟。我想，做一个资料汇编，采撷各家言论之精要，对新生学习应该会有较大帮助，至少一可以减少他们上网盲目搜索的时间，二可以给他们构造一个相对完整的知识体系。

感谢 ACM-ICPC 先辈们作出的杰出工作和贡献，使得我们这些后继者们可以站在巨人的肩膀上前行。

感谢校集训队各位队员的无私、真诚和抱负的崇高使命感、责任感，能够任劳任怨、以苦为乐的做好这件我校的开创性工作。

唐远新

2012 年 10 月

编写说明

本资料为哈尔滨理工大学 ACM-ICPC 集训队自编自用的内部资料，不作为商业销售目的，也不用于商业培训，因此请各参与学习的同学不要外传。

本分册大纲由黄李龙编写，内容由曹振海、陈禹等分别编写和校核。

本分册内容大部分采编自各 OJ、互联网和部分书籍。在此，对所有引用文献和试题的原作者表示诚挚的谢意！

由于时间仓促，本资料难免存在表述不当和错误之处，格式也不是很规范，请各位同学对发现的错误或不当之处向acm@hrbust.edu.cn邮箱反馈，以便尽快完善本文档。在此对各位同学的积极参与表示感谢！

哈尔滨理工大学在线评测系统（Hrbust-OJ）网址：http://acm.hrbust.edu.cn，欢迎各位同学积极参与AC。

国内部分知名 OJ：

杭州电子科技大学：http://acm.hdu.edu.cn

北京大学：http://poj.org

浙江大学：http://acm.zju.edu.cn

以下百度空间列出了比较全的国内外知名 OJ:
http://hi.baidu.com/leo_xxx/item/6719a5ffe25755713c198b50

哈尔滨理工大学 ACM-ICPC 集训队

2012 年 12 月

目　录

序…… I
编写说明 II

第 6 章 数论 6
6.1 欧几里德算法 6
6.1.1 基本原理 6
6.1.2 解题思路 6
6.1.3 模板代码 6
6.1.4 经典题目 6
6.2 扩展欧几里德算法 8
6.2.1 基本原理 8
6.2.2 解题思路 8
6.2.3 模板代码 8
6.2.4 经典题目 9
6.2.5 扩展变型 12
6.3 筛素数 12
6.3.1 基本原理 12
6.3.2 解题思路 12
6.3.3 模板代码 12
6.3.4 经典题目 13
6.4 模线性方程 14
6.4.1 基本原理 15
6.4.2 解题思路 15
6.4.3 模板代码 15
6.4.4 经典题目 15
6.4.5 扩展变型 16
6.5 求解线性同余方程组 16
6.5.1 基本原理 16
6.5.2 解题思路 17
6.5.3 模板代码 17
6.5.4 经典题目 17
6.5.5 扩展变型 19
6.6 随机素数测试(Miller-Rabin) 19
6.6.1 基本原理 19
6.6.2 解题思路 19
6.6.3 模板代码 19
6.6.4 经典题目 20
6.7 素因子快速分解 21
6.7.1 基本原理 21
6.7.2 解题思路 21
6.7.3 模板代码 21
6.7.4 经典题目 22
6.7.5 扩展变型 24

6.8 整数因子分解算法(Pollard-Rho)....24
6.8.1 基本原理....24
6.8.2 解题思路....24
6.8.3 模板代码....25
6.8.4 经典题目....25
6.8.5 扩展变型....28
6.9 欧拉函数....28
6.9.1 基本原理....28
6.9.2 解题思路....28
6.9.3 模板代码....28
6.9.4 经典题目....28
6.9.5 扩展变型....29
6.10 高斯消元....29
6.10.1 基本原理....29
6.10.2 解题思路....29
6.10.3 模板代码....29
6.10.4 经典题目....32
6.10.5 扩展变型....36
第 7 章 组合数学....37
7.1 母函数....37
7.1.1 基本原理....37
7.1.2 解题思路....37
7.1.3 模板代码....38
7.1.4 经典题目....38
7.1.5 扩展变型....39
7.2 置换群....39
7.2.1 基本原理....39
7.2.2 解题思路....40
7.2.3 模板代码....40
7.2.4 经典题目....40
7.2.5 扩展变型....42
7.3 排列组合....42
7.3.1 基本原理....42
7.3.2 解题思路....43
7.3.3 模板代码....43
7.3.4 经典题目....43
7.4 卡塔兰数(Catalan)....45
7.4.1 基本原理....45
7.4.2 解题思路....45
7.4.3 模板代码....46
7.4.4 经典题目....46
7.5 容斥原理....47
7.5.1 基本原理....47
7.5.2 解题思路....47
7.5.3 模板代码....47

7.5.4 经典题目 47
7.6 Burnside定理 50
7.6.1 基本原理 50
7.6.2 解题思路 51
7.6.3 模板代码 51
7.6.4 经典题目 52
7.7 Polya计数 55
7.7.1 基本原理 55
7.7.2 解题思路 55
7.7.3 模板代码 55
7.7.4 经典题目 55

第6章 数论

6.1 欧几里德算法

参考文献：

《ACM—ICPC 程序设计系列<数论及应用>》——哈尔滨工业大学出版社

《初等数论》第二版（潘承洞，潘承彪著）北京大学出版社

最大公约数-百度百科

扩展阅读：

编写：曹振海　　校核：陈禹

6.1.1 基本原理

欧几里得算法又称为辗转相除法，设两个数 a,b 则 a,b 的最大公约 gcd(a,b)=gcd(b,a%b)

不妨设 a>=b,c=gcd(a,b),a=kc,b=jc,则 k,j 互素（否则 c 不是 a,b 的最大公约数），则设 r=a%b 则 a=mb+r, 则 r=a-mb=kc-mjc=(k-mj)c, 因为 k,j 互素，则 k-mj 与 j 互素 gcd(a,b)=gcd(b,a%b)

6.1.2 解题思路

一般很少有题目只要求最大公约数，但是通常很多题会用到求最大公约数的过程，比如判断两个数是否互素（最大公约数为 1），这个时候欧几里得算法暨辗转相除法就变成了很得力的工具，因为每一步都是取模，保证了数据减小的速度特别快。能够在很短的时间内求出两个数的最大公约数。

6.1.3 模板代码

//非递归版，C 语言实现

```
int gcd(int a,int b)
{
    if(!a)
     return b;
    int c;
    while(b)
    {
        c=b;
        b=a%b;
        a=c;
    }
    return a;
}
```

//递归版，C 语言实现

```
int gcd(int a,int b)
{
 if(b==0)
     return a;
 return gcd(b,a%b);
}
```

6.1.4 经典题目

6.1.4.1 题目 1

1．题目出处/来源

哈理工OJ-1328 相等的最小公倍数

2．题目描述

定义 An 为 1，2，…，n 的最小公倍数，例如，A1=1，A2=2，A3=6，A4=12，A5=60，A6=60。

请你判断对于给出的任意整数 n，An 是否等于 An–1。

如果 An 等于 An-1 则输出 YES 否则输出 NO。

3．分析

由最小公倍数的定义我们可以知道，如果 An=An-1 则 An-1 可以被 n 整除，首先，对于一个数 n 如果是素数，那么 An 不等于 An-1,其次，我们分析 n，如果对于小于 n 的每一对因子即 n=a*b(a<n 且 b<n),如果 a,b 不互素，那么 gcd(a,b)>1,则 lcm(a,b)=n/gcd(a,b)<n(lcm 表示最小公倍数)，那么很显然，如果 n 的每一对因子都是不互素的，则 n 不能整除 An-1,否则可以整除 An-1,因为 n 的因子肯定小于等于 n-1，所以其每一对因子的最小公倍数肯定可以整除 An-1,所以这道题就 0 变成了判断 n 是否有互素的一对因子，所以我们只要枚举 n 的每一对因子，然后计算其最大公约数是否为 1,如果有一对互素的因子则输出 YES，否则输出 NO

4．代码（包含必要注释，采用最适宜阅读的 Courier New 字体，小五号，间距为固定值 12 磅）

```
#include<stdio.h>
#include<math.h>
#include<string.h>
int gcd(int a,int b)//求最大公约数部分
{
    if(b==0)
    return a;
    return gcd(b,a%b);
}
int main()
{
    int n;
    int t;
    int i,j;
    scanf("%d",&t);
    while(t--)
    {
        bool p=true;
        scanf("%d",&n);
        if(n==2)//特殊情况的判断
        {
            printf("NO\n");
            continue;
        }
        int temp=(int)sqrt((double)n);
        for(i=2;i<=temp;i++)//判断 n 是否是素数
        {
            if(n%i==0)
            break;
        }
        if(i==temp+1)//是素数直接输出 NO
        {
            printf("NO\n");
            continue;
        }
        for(i=2;i<=temp;i++)//枚举 n 的每一对因子是否互素
        {
            if(n%i==0)
            {
                int q=n/i;
                if(gcd(q,i)==1)
                {
                    p=false;
```

```
                    break;
                }
            }
        }
        if(p)
        printf("NO\n");
        else
        printf("YES\n");
    }
    return 0;
}
```

5．思考与扩展：在辗转相除法执行之前是否一定要保证 a>=b，为什么？

6.2 扩展欧几里德算法

参考文献：

扩展欧几里得-百度百科

扩展欧几里得解线性方程

扩展阅读：

编写：曹振海　　校核：陈禹

6.2.1 基本原理

设 a 和 b 不全为 0，则存在整数 x,y 使得 gcd(a,b)=xa+yb

对于辗转相除法的最后一项

此时 b=0,则 gcd(a,b)=1*a+0*b,因为 gcd(a,b)=gcd(b,a%b)则有 x*a+y*b=x1*b+y1*(a%b)

将等式右边变形，b*x1+(a%b)*y1=b*x1+(a-(a/b)*b)*y1=a*y1+b*(x1-(a/b)*y1)

则，x=y1,y=x1-(a/b)*y1 则可由后向前迭代得到 x,y

6.2.2 解题思路

对于扩展欧几里德定理的题，一般都需要进行一定的推导之后得到一个形式为 xa+yb=c 的方程，然后根据 c 确定解是否存在，如果 c 可以被 gcd(a,b)整除，那么方程有解，否则方程无解。而且所得的解释不唯一的，对于一组解 x0,y0 则其所有解可以表示为 x=x0+b/gcd(a,b)*t,y-y0-a/gcd(a,b)*t,t=0,+1,+2……一般会要求找出 x 或者 y 的最小正整数解，这个时候需要做一些调整。

6.2.3 模板代码

```
int exgcd(int a,int b,int &x,int &y)
{
    if(b==0)
    {
        x=1;
        y=0;
        return a;
    }
    int d=exgcd(b,a%b,x,y);
    int t=x;
    x=y;
    y=t-a/b*y;
    return d;
}
```

6.2.4 经典题目

6.2.4.1 题目 1

1．题目出处/来源

POJ-1061 青蛙的约会

2．题目描述

两只青蛙在网上相识了，它们聊得很开心，于是觉得很有必要见一面。它们很高兴地发现它们住在同一条纬度线上，于是它们约定各自朝西跳，直到碰面为止。可是它们出发之前忘记了一件很重要的事情，既没有问清楚对方的特征，也没有约定见面的具体位置。不过青蛙们都是很乐观的，它们觉得只要一直朝着某个方向跳下去，总能碰到对方的。但是除非这两只青蛙在同一时间跳到同一点上，不然是永远都不可能碰面的。为了帮助这两只乐观的青蛙，你被要求写一个程序来判断这两只青蛙是否能够碰面，会在什么时候碰面。

我们把这两只青蛙分别叫做青蛙 A 和青蛙 B，并且规定纬度线上东经 0 度处为原点，由东往西为正方向，单位长度 1 米，这样我们就得到了一条首尾相接的数轴。设青蛙 A 的出发点坐标是 x，青蛙 B 的出发点坐标是 y。青蛙 A 一次能跳 m 米，青蛙 B 一次能跳 n 米，两只青蛙跳一次所花费的时间相同。纬度线总长 L 米。现在要你求出它们跳了几次以后才会碰面。

3．分析

假设跳了 T 次以后，青蛙 1 的坐标便是 x+m*T,青蛙 2 的坐标为 y+n*T。它们能够相遇的情况为（x+m*T）-(y+n*T)==P*L，其中 P 为某一个整数，变形一下

得到(n-m)*T-P*L==x-y 我们设 a=(n-m),b=L,c=x-y,T=x,P=y.于是便得到 ax+by==c，直接利用欧几里得扩展定理可以得到一组 x,y 但是这组 x,y 不一定是符合条件的最优解，首先,当 gcd(a,b)不能整除 c 的时候是无解的，当 c 能整除 gcd(a,b)时，把 x 和 y 都要变为原来的 c/gcd(a,b)倍，我们知道它的通解为 x0+b/gcd(a,b)*t 要保证这个解是不小于零的最小的数，我们先计算当 x0+b/gcd(a,b)*t=0 时的 t 值，此时的 t 记为 t0=-x0/b/gcd(a,b)（整数除法），代入 t0 如果得到的 x 小于零再加上一个 b/gcd(a,b)就可以了。

4．代码（包含必要注释，采用最适宜阅读的 Courier New 字体，小五号，间距为固定值 12 磅）

```
#include<iostream>
#include<string>
#include<cmath>
#include<algorithm>
usingnamespace std;
__int64 x,y,a,b,c,d;
__int64 n,m,X,Y,L;

__int64 gcd(__int64 a,__int64 b)
{
    __int64 t,d;
    if(b==0)
    {
        x=1;
        y=0;
        return a;
    }
    d=gcd(b,a%b);
    t=x;
    x=y;
    y=t-(a/b)*y;
```

```
    return d;
}

int main()
{
    while(scanf("%I64d%I64d%I64d%I64d%I64d",&X,&Y,&m,&n,&L)==5)
    {
        a=n-m;//确定 a 和 b 的值
        b=L;
        c=X-Y;
        d=gcd(a,b);
        if(c%d!=0)//无解的情况
        {
            printf("Impossible\n");
            continue;
        }
        x=x*(c/d);
        y=y*(c/d);

        /*通解:
        x1=x+b/d*t;
        y1=y-a/d*t;
        t 为任意整数
        */
        //找最小的 x1，即求 x+b/d*t 最小，那么只有 t 为某一个数时才最小
        //先确定式子等于 0 时的 t 值，然后进行调整
        __int64 k=x*d/b;
        k=x-k*b/d;
        if(k<0)
            k+=b/d;
        printf("%I64d\n",k);
    }
    return0;
}
```

6.2.4.2 题目 2

1. 题目出处/来源

POJ-2142The Balance

2. 题目描述

一个人有一种天平，这种天平只有两种重量的砝码 a 和 b，现在要称出重量为 c 的物品，问你至少需要多少 a 和 b，答案需要满足 a 的数量加上 b 的数量和最小，并且他们的重量和也要最小。（两个盘都可以放砝码）

3. 分析

假设 a 砝码我们用了 x 个，b 砝码我们用了 y 个。那么天平平衡时，就应该满足 ax+by==c。x,y 为正时表示放在和 c 物品的另一边，为负时表示放在 c 物品的同一边。于是题意就变成了求|x|+|y|的最小值了。x 和 y 是不定式 ax+by==c 的解。刚刚上面已经提到了关于 x,y 的所以解的同式，即

x=x0+b/d*t

y=y0-a/d*t

穷举所有解，取|x|+|y|最小的？显然是行不通的，仔细分析：

|x|+|y|==|x0+b/d*t|+|y0-a/d*t|，我们规定 a>b(如果 a<b，我们便交换 a b)，从这个式子中，我们可以轻易的发现：|x0+b/d*t|是单调递增的，|y0-a/d*t|是单调递减的，而由于我们规定了 a>b，那么减的斜率边要大于增的斜率，于是整个函数减少的要比增加的快，但是由于绝对值的符号的作用，最终函数还是递增的。也就是说，函数是凹的，先减小，再增

大。那么什么时候最小呢？很显然是 y0-a/d*t==0 的时候，于是我们的最小值|x|+|y|也一定是在 t=y0*d/a 附近了，只要在附近枚举几个值就能找到最优解了。

4. 代码

```
#include<iostream>
#include<cstdio>
#include<cstring>
#define inf 0x7fffffff
using namespace std;
int Abs(int a)
{
    return a<0?-a:a;
}
int min(int a,int b)
{
    return a<b?a:b;
}
int max(int a,int b)
{
    return a>b?a:b;
}
int exgcd(int a,int b,int &x,int &y)
{
    if(b==0)
    {
        x=1;
        y=0;
        return a;
    }
    int d=exgcd(b,a%b,x,y);
    int t=x;
    x=y;
    y=t-(a/b)*y;
    return d;
}
int main()
{
    int a,b,c,x,y,d,x1,y1,ansx,ansy,i;
    bool sf;
    while(scanf("%d%d%d",&a,&b,&c)&&(a||b||c))
    {
        sf=false;
        if(a<b)//把减小速度快的作为 a
        {
            sf=true;
            swap(a,b);
        }
        d=exgcd(a,b,x,y);
        x*=(c/d);
        y*=(c/d);
        int t=y*d/a;
        ansx=Abs(x+t*b/d);
        ansy=Abs(y-t*a/d);
        for(i=1;i<5;i++)
        {
            x1=x+(t+i)*b/d;//分别在 t 的附近找四个值
            y1=y-(t+i)*a/d;
            x1=Abs(x1);
            y1=Abs(y1);
```

```
            if(x1+y1<ansx+ansy||(x1+y1==ansx+ansy&&x1*a+y1*b<ansx*a+ansy*b))
            {
               ansx=x1;
               ansy=y1;
            }
            x1=x+(t-i)*b/d;
            y1=y-(t-i)*a/d;
            x1=Abs(x1);
            y1=Abs(y1);
            if(x1+y1<ansx+ansy||(x1+y1==ansx+ansy&&x1*a+y1*b<ansx*a+ansy*b))
            {
               ansx=x1;
               ansy=y1;
            }
         }
         if(sf)
         swap(ansx,ansy);
         printf("%d %d\n",ansx,ansy);
      }
      return 0;
   }
```

6.2.5 扩展变型

POJ-2891Strange Way to Express Integers

6.3 筛素数

参考文献：

筛法求素数

扩展阅读：

编写：曹振海　　校核：陈禹

6.3.1 基本原理

筛素数的基本方法是用来筛选出一定范围内的素数

素数筛法的基本原理，利用的是素数 p 只有 1 和 p 这两个约数，并且一个数的约数一定不大于本身，素数筛法的过程：把从 1 开始的、某一范围内的正整数从小到大顺序排列， 1 不是素数，首先把它筛掉。剩下的数中选择最小的数是素数，然后去掉它的倍数。依次类推，直到筛子为空时结束。

6.3.2 解题思路

素数筛法经常作为一道题的一部分用来打一定范围内素数表，然后利用素数表作为基础解题。

6.3.3 模板代码

//素数筛法.cpp

```
bool isprime[N];//N 表示范围
int prime[N],cnt;
void f()
{
   int i,j;
  cnt=0;
   memset(isprime,true,sizeof(isprime));
   isprime[1]=false;
   for(i=2;i<=N;i++)
   {
```

```
        if(isprime[i])
        {
            prime[cnt++]=i;//记录素数
            for(j=i*i;j<=N;j+=i)//因为小于 i 的所有的倍数都被筛过，所以直接从 i*i 开始，从这里也可以看出，筛素数时到 N^0.5 就可以了
            isprime[j]=false;
        }
    }
}
```

6.3.4 经典题目

6.3.4.1 题目 1

1．题目出处/来源

POJ-2689Prime Distance

2．题目描述

给出一个区间[L,U]找出这个区间内相邻的距离最近的两个素数和距离最远的两个素数

1<=L<U<=2,147,483,647 区间长度不超过 1,000,000

3．分析

因为给出的 U 的范围达到了 int 的最大，所以不能直接打出 1-U 的素数表，我们知道，利用素数筛时是把所有的合数都筛掉了，那么我们也想办法把 L-U 内的合数都筛掉就可以了，那么只要用 √U 内的素数去筛掉 L-U 的合数就行了

4．代码（包含必要注释，采用最适宜阅读的 Courier New 字体，小五号，间距为固定值 12 磅）

```
#include<iostream>
#include<cstring>
#include<cstdio>
#define N 50000
#define len 1000000
#define inf 0x7fffffff
using namespace std;
bool isprime[N+5];
long long prime[N],cnt;
bool res[len+5];
void init()//筛选 50000 内的素数，要注意 i 比较大时会容易超 int 的范围
{
    long long i,j;
    cnt=0;
    memset(isprime,true,sizeof(isprime));
    for(i=2;i<=N;i++)
    {
        if(isprime[i])
        {
            prime[cnt++]=i;
            if(i*i<=N)
            {
                for(j=i*i;j<=N;j+=i)
                {
                    isprime[j]=false;
                }
            }
        }
    }
}
int main()
```

```
    {
        long long L,U,i,j,k,min,max,s,t;
        init();
        while(scanf("%lld%lld",&L,&U)!=EOF)
        {
            memset(res,0,sizeof(res));
            for(i=0;i<cnt;i++)//筛选区间内的素数
            {
                s=(L-1)/prime[i]+1;
                t=U/prime[i];
                for(j=s;j<=t;j++)
                if(j>1)
                res[j*prime[i]-L]=true;
            }
            k=-1,min=inf,max=-1;
            long long dis,m1,m2;
            for(i=0;i<=U-L;i++)//最优值求解
            {
                if(!res[i])
                {
                    if(k!=-1)
                    {
                        dis=i-k;
                        if(dis>max)
                        {
                            max=dis;
                            m1=i;
                        }
                        if(dis<min)
                        {
                            min=dis;
                            m2=i;
                        }
                    }
                    if(i+L!=1)//注意 1 的时候特殊判断 1 不是素数
                    k=i;
                }
            }
            if(max==-1)//无解的情况
            {
                printf("There are no adjacent primes.\n");
            }
            else
            {
                printf("%lld,%lld  are  closest,  %lld,%lld  are  most  distant.\n",m2-
min+L,m2+L,m1-max+L,m1+L);
            }
        }
        return 0;
    }
```

6.4 模线性方程

参考文献：

ACM-ICPC 程序设计系列 数论及其应用

扩展阅读：

用模线性方程解决逆元问题

http://wenku.baidu.com/view/f0894659be23482fb4da4c51.html

编写：陈禹　　校核：曹振海

6.4.1 基本原理

解决形如 ax=b(mod n)

推论 1：方程 ax=b(mod n)对于未知量 x 有解，当且仅当 gcd(a,n) | b。

推论 2：方程 ax=b(mod n)或者对模 n 有 d 个不同的解，其中 d=gcd(a,n)，或者无解。

定理 1：设 d=gcd(a,n)，假定对整数 x 和 y 满足 d=ax+by(比如用扩展 Euclid 算法求出的一组解)。如果 d | b，则方程 ax=b(modn)有一个解 x0 满足 x0=x*(b/d) mod n 。特别的设 e=x0+n，方程 ax=b(mod n)的最小整数解 x1=e mod(n/d)，最大整数解 x2=x1+(d-1)*(n/d)。

定理 2：假设方程 ax=b(mod n)有解，且 x0 是方程的任意一个解，则该方程对模 n 恰有 d 个不同的解(d=gcd(a,n))，分别为：xi=x0+i*(n/d) mod n 。

以上定理的具体证明见《算法导论》

6.4.2 解题思路

其中 a,b,n 是已知的（n>0），要求解出满足上式的对模 n 的 x 值。满足同余方程的 x 可能有多个，也可能一个都没有，上述模线性方程也称为一次同余方程。

例如：57x=7（mod 11）有一个解 x=9，而 9x=7（mod 6）无解。

解：模线性方程 ax=b（mod n）的步骤如下：

1. 求 d=gcd（a,n）
2. 若 d 不是 b 的因数，则 ax=b(mod n)无解，结束；否则转（3）
3. 求 x0,y0,是 a*x0+n*y0=d;
4. 由于 d 是 b 的因数，将 a*x0+n*y0=d 改写，得 a(x0*(b/d))+n*(y0*(b/d))=b，于是 ax+ny=b 的一个特解为 x=x0*(b/d),y=y0*(b/d)。
5. x0*(b/d)是 ax=(mod n)的一个特解，由此的 ax=b(mod n)的所有解（共 d 个）为：x=(x0*(b/d)+ i*(n/d)) (mod n), i=0,1,2,3,4.……d-1.。

6.4.3 模板代码

sol 数组储存值为对于 n 的所有剩余系(从小到大)值为非负,数量为 gcd(a,n)。

此 sol 数组在绝大数题中不需要,可以根据题意决定是否使用。

```
int modeq(int a, int b, int n, int sol[]) {// ! n > 0
   int e, i, d, x, y, f;
   d = extgcd(a, n, x, y);//注意此处可能出现负值
   if(b % d) return -1;
   else {
      f = n / d < 0 ? ( -1 * n / d) : n / d;//处理负值情况
      e = (x * (b / d) % f + f) % f;
      d = d < 0 ? - 1 * d : d;
      for(i = 0; i < d; ++i)
      sol[i] = e + i * f;
   }
}
```

6.4.4 经典题目

6.4.4.1 题目 1

1．题目出处/来源

POJ 2115 C Looooops

2．题目描述

给你一个变量，变量初始值 a，终止值 b，每循环一遍加 c，问一共循环几遍终止，结果 mod2^k.如果无法终止则输出 FOREVER。

3．分析

根据题意原题可化成 c * x = b - a mod (2 ^ k)，解这个模线性方程，输出最小正解即可。

4．代码（包含必要注释，采用最适宜阅读的 Courier New 字体，小五号，间距为固定值 12 磅）

```
#include<stdio.h>
typedef long long ll;
ll extgcd(ll a, ll b,ll & x, ll & y) {//扩展欧几里德
   if (b == 0) {x = 1; y = 0; return a;}
   ll d = extgcd(b, a % b, x, y);
   ll t = x; x = y; y = t - a / b * y;
   return d;
}
ll modeq(ll a, ll b, ll n) {//求解模线性方程
   ll e, i, d, x, y, f;
   d = extgcd(a, n, x, y);
   if (b % d) return -1;
   else {
   f = n / d < 0 ? (-1 * n / d) : n / d;
   e = (x * (b / d) % f + f) % f;
   d = d < 0 ? -1 * d : d;
   return e;
   }
}
int main() {
   ll a, b, c, k, m, tmp, flag;
   while(scanf("%I64d%I64d%I64d%I64d", &a, &b, &c, &k), a + b + c + k != 0) {
      m = 1LL << k;
      tmp = b - a;
      flag = modeq(c, tmp, m);
      if(flag == -1)
         printf("FOREVER\n");
      else
         printf("%I64d\n",flag);
   }
   return 0;
}
```

6.4.4.2 题目 2

6.4.5 扩展变型

6.5 求解线性同余方程组

参考文献：

ACM-ICPC 程序设计系列 数论及其应用

扩展阅读：

编写：陈禹　　校核：曹振海

6.5.1 基本原理

对于模线性方程组，可以进行方程组合并，求出合并后的方程的解，这样就可以很快的推出方程的最终解，不管这样的方程有多少个，都可以俩俩解决，求得方程组的最终解，所以仅以

x=b1(mod m1) (1);

x=b2(mod m2) (2);

俩个方程构成的方程组进行讲解。

令 m=[m1,m2].

首先，此次方程组有解的充分必要条件是(m1,m2)|(b1,b2)，此时方程仅有一个小于 m 的非负整数解，利用扩展欧几里德算法解出来。

式(1)等价于 x=b1+m1y1;式(2)等价于 x=b2+m2y2;

联立可得 b1+m2y1=b2+m2y2,即 m2y2-m1y1=b1-b2;

根据之前所学模线性方程的方法，很容易得到此方程的解有 y2，因此小于 m 的非负整数解即为(b2+m2y2)%m。

假设同余方程组为

$$x=r1(\bmod a1);$$
$$x=r2(\bmod a2);$$
$$x=r3(\bmod a3);$$
$$\vdots$$
$$x=rn(\bmod an);$$

依次俩俩求解即可

6.5.2 解题思路

根据题意列出线性同余方程组，套用模版得出答案即可

6.5.3 模板代码

形如 x%a[i]=r[i];

返回值为-1 代表方程无解,否则则返回最小非负解.

```
int solve(int a[], int r[], int n) {
   int d, c ,i, x, y, t;
   for(i=1;i<n;i++) {
      c = r[i] - r[i - 1];
      d = extgcd(a[i - 1], a[i], x, y);
      if(c % d != 0) return -1;
      t = a[i] / d;
      x = (x * (c / d) % t + t) % t;
      r[i] = a[i - 1] * x + r[i - 1];
      a[i] = a[i - 1] * (a[i] / d);
   }
   return r[n - 1];
}
```

6.5.4 经典题目

6.5.4.1 题目 1

1．题目出处/来源

HDU 1573 X 问题

2．题目描述

求在小于等于 N 的正整数中有多少个 X 满足：X mod a[0] = b[0], X mod a[1] = b[1], X mod a[2] = b[2], …, X mod a[i] = b[i], … (0 < a[i] <= 10)。

3．分析

判断同余方程组在一定范围内解的个数的问题，另 b 数组中所有的最小公倍数是 lcm，方程在 lcm 范围内的非负整数解是 a，则有 a+lcm*x<=N,即解的个数为(N-a)/lcm+1，若 a==0 则解的个数减一即可。因为题目要求是解是正整数。

4．代码（包含必要注释，采用最适宜阅读的 Courier New 字体，小五号，间距为固定值 12 磅）

```
#include<stdio.h>
typedef long long ll;
ll gcd(ll x, ll y) {
   if (!x || !y) return x > y ? x : y;
   for(ll t; t = x % y; x = y, y = t);
   return y;
}
ll extgcd(ll a, ll b,ll & x, ll & y) {
   if (b == 0) {
     x = 1; y = 0;
     return a;
   }
   ll d = extgcd(b, a % b, x, y);
   ll t = x; x = y; y = t - a / b * y;
   return d;
}
ll modeqset(ll a[],ll r[],ll n) {
   ll d, c, i, x, y, t;
   for(i = 1; i < n; ++i) {
      c = r[i] - r[i - 1];
      d = extgcd(a[i - 1], a[i], x, y);
      if(c % d != 0) return -1;
      t = a[i] / d;
      x = (x * (c / d) % t + t) % t;
      r[i] = a[i - 1] * x + r[i - 1];
      a[i] = a[i - 1] * (a[i] / d);
   }
   return r[n - 1];
}
int main() {
   ll a[30], b[30], m, tmp, flag, solve[10000], lcm, num;
   int t, n, i;
   scanf("%d", &t);
   while(t--) {
      scanf("%d%I64d", &n, &m);
      lcm=1;
      for(i = 0;i < m; ++i) {
       scanf("%I64d", &a[i]);
       lcm = lcm / gcd(lcm, a[i]) * a[i];
      }
      for(i = 0; i < m; ++i)
       scanf("%I64d", &b[i]);
      flag=modeqset(a,b,m);
      if(flag == -1 || flag > n)
       printf("0\n");
      else {
       num = (n - flag) / lcm + 1;
       if(flag == 0)
          num--;
       printf("%I64d\n",num);
      }
   }
   return 0;
}
```

5. 思考与扩展：可以借鉴北大培训教材中做法。

6.5.4.2 题目 2

6.5.5 扩展变型

6.6 随机素数测试(Miller-Rabin)

参考文献：

ACM-ICPC 程序设计系列 数论及其应用

扩展阅读：

编写：陈禹　　校核：曹振海

6.6.1 基本原理

费尔马小定理:如果 p 是一个素数,且 0<a<p,则 a^(p-1)%p=1.

利用费尔马小定理,对于给定的整数 n,可以设计素数判定算法,通过 计算 d=a^(n-1)%n 来判断 n 的素性,当 d!=1 时,n 肯定不是素数,当 d=1 时,n 很可能是素数.

二次探测定理:如果 p 是一个素数,且 0<x<p,则方程 x^2%p=1 的解为:x=1 或 x=p-1.

利用二次探测定理,可以再利用费尔马小定理计算 a^(n-1)%n 的过程 中增加对整数 n 的二次探测,一旦发现违背二次探测条件,即得出 n 不是素数的结论.

如果 n 是素数,则(n-1)必是偶数,因此可令(n-1)=m*(2^q),其中 m 是正奇数(若 n 是偶数,则上面的 m*(2^q)一定可以分解成一个正奇数乘以 2 的 k 次方的形式),q 是非负整数,考察下面的测试:

序列:a^m%n; a^(2m)%n; a^(4m)%n; …… ;a^(m*2^q)%n

把上述测试序列叫做 Miller 测试,关于 Miller 测试,有下面的定理:

定理:若 n 是素数,a 是小于 n 的正整数,则 n 对以 a 为基的 Miller 测试,结果为真.

Miller 测试进行 k 次,将合数当成素数处理的错误概率最多不会超过 4^(-k).

Miller_Rabin 测试 T 次时，它产生一个假的素数所花费的时间不超过 1/4^T

6.6.2 解题思路

适用于判断大素数。

6.6.3 模板代码

```
typedef long long ll;
#define times 10//伪素数测试次数,算法出错概率<=4^(-times)
int witness(ll a, ll n)
{
    ll m = n - 1;
    int j = 0;
    while(!(m & 1)) {
        j++;
        m >>= 1;
    }
    long long x = quickmod(a, m, n);
    if(x == 1 || x == n-1) return 0;
    while(j--) {
        x = multi(x, x, n);
        if(x == n-1)
            return 0;
    }
    return 1;
}
int miller_rabin(ll n)
{
    int i;
```

```
    if(n == 1) return 0;
    if(n == 2) return 1;
    if(n % 2 == 0) return 0;
    for(i = 1; i <= times; ++i) {
        ll a = random(n - 2) + 1;
        if(witness(a, n))
        return 0;
    }
    return 1;
}
```

6.6.4 经典题目

1．题目出处/来源

Is it a prime ?(zjut 1517)

2．题目描述

给一个数判断 n 是否为素数。

3．分析

由于 n 的数值比较大直接套用 Miller-Rabin 素数测试即可。

4．代码

```
#include<stdio.h>
#include<stdlib.h>
#define C 201
#define times 5
typedef  long long ll;
ll random(ll n)
{
    return (ll)((double)rand() / RAND_MAX * n + 0.5);
}
ll multi(ll a, ll b, ll n)
{
    ll ret = 0;
    a %= n;
    while(b) {
        if(b & 1)
            ret = (ret + a) % n;
        a = (a << 1) % n;
     b >>= 1;
}
return ret;
}
ll quickmod(ll a, ll b, ll n) {
ll ret = 1;
a %= n;
while(b) {
    if(b & 1)
           ret = multi(ret, a, n);//因为乘法可能超长整型范围，所以这里要将乘法换成加法。
     a = multi(a ,a, n);
     b >>= 1;
}
return ret;
}
int witness(ll a, ll n)
{
    ll m = n - 1;
    int j = 0;
    while(!(m & 1)) {
        j++;
```

```
        m >>= 1;
    }
    long long x = quickmod(a, m, n);
    if(x == 1 || x == n-1) return 0;
    while(j--) {
        x = multi(x, x, n);
        if(x == n-1) return 0;
    }
    return 1;
}
int miller_rabin(ll n) {
    int i;
    if(n == 1) return 0;
    if(n == 2) return 1;
    if(n % 2 == 0) return 0;
    for(i = 1; i <= times; ++i) {
        ll a = random(n - 2) + 1;
        if(witness(a, n)) return 0;
    }
    return 1;
}
int main() {
    ll n;
    while(scanf("%I64d", &n) != EOF) {
        if(miller_rabin(n))
            printf("Yes\n");
        else
            printf("No\n");
    }
    return 0;
}
```

6.7 素因子快速分解

参考文献：

ACM-ICPC 程序设计系列 数论及其应用

扩展阅读：

编写：陈禹　　校核：曹振海

6.7.1 基本原理

试除法是整数分解算法中最简单和最容易理解的算法。

给定一个合数 n（这里，n 是待分解的整数），试除法看成是用小于等于根号 n 的每个素数去试除待分解的整数。如果找到一个数能够整除除尽，这个数就是待分解整数的因子。

6.7.2 解题思路

常用于对整数进行素因子分解，利用素因子指数和素因子来解题。适用于小整数

6.7.3 模板代码

```
typedef long long ll;
#define N 1000010
ll prime[N], nprime, factor[N], numfactor[N], ct;
void makeprime() {//打 1~N 的素数表
    int i, j, temp;
    nprime = 0;
    memset(isprime, 1, sizeof(isprime));
```

```
        isprime[1] = 0;
        temp = sqrt(N + 0.0);
        for(i = 2;i <= temp; ++i) {
            if(isprime[i]) {
                ++nprime;
                prime[nprime] = i;
                for(j=i+i; j < N; j += i) {
                    isprime[j] = 0;
                }
            }
        }
    }
    void divide(ll n) {//进行素数分解
        int i;
        int temp = sqrt(n + 0.0);
        ct=0;
        memset(numfactor, 0, sizeof(numfactor));
        for(i = 1; i <= nprime; ++i) {
            if(prime[i] > temp) break;
            if(n % prime[i] == 0) {
                factor[++ct]=prime[i];
                while(n % prime[i] == 0) {
                    numfactor[ct]++;
                    n /= prime[i];
                }
            }
        }
        if(n != 1) {
            factor[++ct] = n;
            numfactor[ct]++;
        }
    }
```

6.7.4 经典题目

1．题目出处/来源

hrbust oj 1344 Unit Fraction

2．题目描述

单位分数是一个特殊的分数。例如,1/2,1/3 等都是单位分数,作为他们的分子是 1 并且分母是大于 1 的正整数。

我们想知道一个分数 n/m 可以多少种两个单位分数。例如：

2/3 = 1/3 + 1/3 = 1/2 + 1/6;3/4 = 1/2 + 1/4

3．分析

首先任何一个整数都可以分解成若干个质数乘积的形式。

把式子转化一下变为 (mx - n)(my - n) = n^2

可以简单的证明一下 如果有 x 满足

n^2 % (mx - n) == 0 则一定存在对应的 y 使(mx - n)(my - n) = n^n 成立.

设 x <=y , 那么再根据对称性可得, mx - n <= n.

其实就是求有在[1...n]中有多少 mx - n 恰好是 n^2 的因子.

那么我们深搜求出[1...n]中所有的 n^2 的因子,然后一一验证一下就可以了.

4．代码（包含必要注释，采用最适宜阅读的 Courier New 字体，小五号，间距为固定值 12 磅）

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

```
#define N 1000010
typedef long long ll;
ll prime[N],nprime,factor[N],numfactor[N],ct,n,m;
bool isprime[N];
ll gcd(ll x, ll y) {
if (!x || !y) return x > y ? x : y;
for (ll t; t = x % y; x = y, y = t);
return y;
}
void makeprime() {//打 1~N 的素数表
    int i, j, temp;
    nprime = 0;
    memset(isprime, 1, sizeof(isprime));
    isprime[1] = 0;
    temp = sqrt(N + 0.0);
    for(i = 2;i <= temp; i++) {
        if(isprime[i]) {
            nprime++;
            prime[nprime] = i;
            for(j = i + i;j < N;j += i) {
                isprime[j]=0;
            }
        }
    }
}
void divide(ll n) {
    int i;
    int temp = sqrt(n + 0.0);
    ct = 0;
    memset(numfactor, 0, sizeof(numfactor));
    for(i = 1;i <= nprime; i++) {
        if(prime[i] > temp)
            break;
        if(n % prime[i] == 0) {
        factor[++ct] = prime[i];
        while(n % prime[i] == 0) {
            numfactor[ct]++;
            n /= prime[i];
        }
        }
    }
    if(n!=1) {
    factor[++ct]=n;
    numfactor[ct]++;
    }
}
ll factor1[10000000];
int ct1;
void dfs(int cur,ll val) {//深搜出 n^2 的因子
    if(cur == ct + 1) {
       factor1[++ct1] = val;
       return;
    }
    for(int i = 0;i <= 2 * numfactor[cur]; i++) {
        dfs(cur + 1, val * pow(1.0 * factor[cur], 1.0 * i));
    }
}
int judge(ll a,ll b) {//判断 a,b 是否满足作为分母的条件
    if((a + n) % m == 0 && (b + n) % m == 0 && (a + n) / m !=1 && (b + n) / m != 1)
```

```
            return 1;
        else
            return 0;
    }
    int main()
    {
        makeprime();
        ll k,sum;
        int ca=1;
        while(scanf("%I64d%I64d",&m,&n)!=EOF)
        {
            k = gcd(m,n);
            m /= k; n /= k;
            divide(n);
            ct1=0;
            dfs(1, 1);
            sum = 0;
            for(int i = 1; i <= ct1; i++) {//对每个因子进行分解
                if(judge(factor1[i],n*n/factor1[i]))
                sum++;
            }
            if(judge(n, n))
                sum++;
            sum /= 2;
            printf("Scenario #%d\n",ca++);
            if(sum==0)
                printf("No solution\n\n");
            else if(sum==1)
                printf("Only one solution\n\n");
            else
                printf("Find %lld solutions\n\n",sum);
        }
        return 0;
    }
```

6.7.5 扩展变型

6.8 整数因子分解算法(Pollard-Rho)

参考文献：

ACM-ICPC 程序设计系列 数论及其应用

扩展阅读：

http://blog.csdn.net/hot2346/article/details/5454407

编写：陈禹　　校核：曹振海

6.8.1 基本原理

生成俩个整数 a 和 b，计算 p=(a-b,n),直到 P 不为 1 或 a,b 出现循环为止，若 p=n，则 n 为质数，否则 p 为 n 的一个约数，选取一个小的随机数 x1，迭代生成 xi=xi-1^2+k,一般取 k=1，若序列出现循环，则退出。计算 p=gcd(xi-1-xi,n),p=1,返回上一步，直到 p>1 为止；若 p=n，则 n 为素数，否则 p 为 n 的一个约数并递归分解 p 和 n/p。

6.8.2 解题思路

对于比较大的整数分解，试除法失去实用价值，这时就需要 Pollard-Rho 整数分解方法了

6.8.3 模板代码

对比较大的整数试除法等失去实用价值这时采用 pollard rho 整数分解法

pr[]内存的是分解出的质因数

初始化 ct 为 0

```
#define C 201//防止死循环
typedef  long long ll;
ll pr[100000];
int ct;
ll pollard_rho(ll n, int c) {
    ll x, y, d, i = 1, k = 2;
    x = random(n - 1) + 1;
    y = x;
    while(1) {
        ++i;
        x = (multi(x, x, n) + c) % n;
        d = gcd(y - x + n, n);
        if(d > 1 && d < n) return d;
        if(y == x) return n;
        if(i == k) {
            y = x;
            k *= 2;
        }
    }
    return n;
}
void find(ll n, int k) {//深搜找出因子
    if(n == 1) return;
    if(miller_rabin(n)) {
        pr[ct++] = n;
        return;
    }
    ll p = n;
    while(p >= n)
        p=pollard_rho(p,k-1);
    find(p, k);
    find(n / p, k);
}
```

6.8.4 经典题目

1．题目出处/来源

hrbust oj 1521 水神的素数

2．题目描述

最近水神遇到了一个简单的素数问题,就是一个数是不是素数,使他非常困惑他想请集训队的同学帮助他.

3．分析

由于 n 的数值范围比较大，需要用 Miller-Rabin 进行素数测试，如果是素数，则输出 Prime，如果不是素数，就进行 Pollard-Rho 整数分解方法，求出所有的质因数后取其最小值和最大值。此题应注意如果用乘法可能超数据范围，所以应该将乘法转成加法。

4．代码（包含必要注释，采用最适宜阅读的 Courier New 字体，小五号，间距为固定值 12 磅）

```
#include<stdio.h>
#include<stdlib.h>
#define C 201
#define times 5
```

```
typedef  long long ll;
ll pr[100000];
int ct;
ll gcd(ll x, ll y) {
   if(!x||!y)
      return x > y ? x : y;
   for(ll t; t = x % y; x = y, y = t);
      return y;
}
ll random(ll n) {
   return (ll)((double)rand() / RAND_MAX * n + 0.5);
}
ll multi(ll a, ll b, ll n) { //注意范围超整型
   ll ret = 0;
   a %= n;
   while(b) {
      if(b & 1) {
         ret = (ret + a) % n;
         a = (a << 1) % n;
         b >>= 1;
   }
   }
return ret;
}
ll quickmod(ll a, ll b, ll n) {
ll ret = 1;
a %= n;
while(b) {
    if(b & 1)
      ret = multi(ret, a, n);
    a = multi(a, a, n);
    b >>= 1;
}
return ret;
}
int witness(ll a,ll n) {
   ll m = n - 1;
   int j = 0;
   while(!(m & 1)) {
      j++;
      m >>= 1;
   }
   long long x = quickmod(a , m, n);
   if(x == 1 || x == n-1) return 0;
   while(j--) {
      x = multi(x , x, n);
      if(x == n-1) return 0;
   }
   return 1;
}
int miller_rabin(ll n) {
   int i;
   if(n == 1) return 0;
   if(n == 2) return 1;
   if(n % 2 == 0) return 0;
   for(i = 1; i <= times; ++i) {
      ll a = random(n - 2) + 1;
      if(witness(a, n)) return 0;
   }
```

```
    return 1;
}
ll pollard_rho(ll n, int c) {
    ll x, y, d, i = 1, k = 2;
    x = random(n - 1) + 1;
    y = x;
    while(1) {
        i++;
        x = (multi(x, x, n) + c) % n;
        d = gcd(y - x + n, n);
        if(d > 1 && d < n) return d;
        if(y == x) return n;
        if(i == k) {
            y = x;
            k *= 2;
        }
    }
    return n;
}
void find(ll n, int k) {
    if(n == 1) return;
    if(miller_rabin(n)) {
        pr[ct++] = n;
        return;
    }
    ll p = n;
    while(p >= n)
        p = pollard_rho(p, k - 1);
    find(p, k);
    find(n / p, k);
}
int main() {
    int i, t;
    ll n, min, max;
    scanf("%d", &t);
    while(t--) {
        scanf("%lld", &n);
        if(miller_rabin(n))
            printf("Prime\n");
        else {
            ct = 0;
            min = 1LL << 61;
            max = 0;
            find(n, C);
            for(i = 0; i < ct; i++) {
                if(pr[i] < min)
                    min = pr[i];
                if(pr[i] > max)
                    max = pr[i];
            }
            printf("%lld,%lld\n", min, max);
        }
    }
    return 0;
}
```

6.8.5 扩展变型

6.9 欧拉函数

参考文献：
ACM-ICPC 程序设计系列 数论及其应用
扩展阅读：
http://www.cnblogs.com/DreamUp/archive/2010/07/24/1784116.html
编写：陈禹　　校核：曹振海

6.9.1 基本原理

欧拉函数 phi()是指不超过 n 且与 n 互素的正整数的个数

Φ函数的值 通式： Φ(x)=x(1-1/p1)(1-1/p2)(1-1/p3)(1-1/p4)…..(1-1/pn),其中 p1, p2…… pn 为 x 的所有质因数

设 p 是素数 a 是一个正整数 phi(p^a)=p^a-p^a-1;

m 与 n 互素 phi(mn)=phi(m)phi(n);

phi(n)=n*sum(1-1/pi)/*pi 是与 n 的质因子

n 为奇数时 phi(2n)=phi(n);

欧拉定理:对任何俩个互质的正整数 a,m(m>=2) a^phi(m)=1(modm);

费马小定理:当 m 是质数是 a^m-1=1(modm);

6.9.2 解题思路

根据欧拉函数定义来求解一些数论问题

6.9.3 模板代码

递推求法

```
for (i = 1; i <= maxn; i++) phi[i] = i;
for (i = 2; i <= maxn; i += 2) phi[i] /= 2;
for (i = 3; i <= maxn; i += 2) if(phi[i] == i) {
   for (j = i; j <= maxn; j += i)
   phi[j] = phi[j] / i * (i - 1);
}
```

公式求法

```
int euler(int x) {
   int i, res = x;
   for (i = 2;i < (int)sqrt(x * 1.0) + 1; i++)
   if(x % i == 0) {
      res = res / i * (i - 1);
      while(x % i == 0) x /= i; // 保证 i 一定是素数
   }
   if(x > 1)
   res = res / x * (x - 1);
   return res;
}
```

6.9.4 经典题目

1．题目出处/来源

POJ 2478 Farey Sequence

2．题目描述

给定 N，求所有小于等于 N 的 a/b，gcd(a,b)=1（a<b）.

3．分析

一个法雷序列 Fn 中个数的个数就是分别与 2,3,4,5,….n-1,n 互素的数的个数的和，继而就是求从 2 到 n 连续的欧拉函数的和，因为 N 的范围是[2,1000000],可以直接用欧拉函数的方法来求解，在打表预处理前 n 项和即可.

4．代码（包含必要注释，采用最适宜阅读的 Courier New 字体，小五号，间距为固定值 12 磅）

```
#include<stdio.h>
#include<string.h>
#include<math.h>
long long phi[1000001];
#define maxn 1000000
int main()
{
   int i,j;
   for (i = 1; i <= maxn; i++) phi[i] = i;
   for (i = 2; i <= maxn; i += 2) phi[i] /= 2;
   for (i = 3; i <= maxn; i += 2) if(phi[i] == i) {
      for (j = i; j <= maxn; j += i)
      phi[j] = phi[j] / i * (i - 1);
   }
   for(i = 3;i < 1000001; i++)
      phi[i] += phi[i - 1];
   int n;
   while(scanf("%d", &n), n)
      printf("%lld\n", phi[n]);
   return 0;
}
```

6.9.5 扩展变型

6.10 高斯消元

参考文献：

高斯消元法

高斯消元题目

扩展阅读：

编写：曹振海　　校核：陈禹

6.10.1 基本原理

最基本的高斯消元的原理，我们在初中学到二元函数的时候所用的消去法就已经用到了，数学上，高斯消元法，是线性代数中的一个算法，可用来为线性方程组求解，求出矩阵的秩，以及求出可逆方阵的逆矩阵。当用于一个矩阵时，高斯消元法会产生出一个“行梯阵式”。利用矩阵化成的行阶梯型可以方便的得出未知数的解

6.10.2 解题思路

和数论中的其它知识一样，要用高斯消元，一般也会需要一定的推理，得出线性方程组，再利用高斯消元求解。下面给出高斯消元的模板适用于非常一般的线性方程组的问题，而在很多题目中，会涉及到取模，位运算等，那个时候就要善于灵活的改变模板，最重要的是理解原理。

6.10.3 模板代码

//高斯消元.cpp

```
//这个模板在有浮点数解得时候是可以求出浮点数解的
int a[MAXN][MAXN];//增广矩阵
int x[MAXN];//解集
bool free_x[MAXN];//标记是否是不确定的变元


int gcd(int a,int b)
{
    int t;
    while(b!=0)
    {
        t=b;
        b=a%b;
        a=t;
    }
    return a;
}
int lcm(int a,int b)
{
    return a/gcd(a,b)*b;//先除后乘防溢出
}
// 高斯消元法解方程组(Gauss-Jordan elimination).(-2 表示有浮点数解，但无整数解，
//-1 表示无解，0 表示唯一解，大于 0 表示无穷解，并返回自由变元的个数)
//有 equ 个方程，var 个变元。增广矩阵行数为 equ,分别为 0 到 equ-1,列数为 var+1,分别为 0 到 var.
int Gauss(int equ,int var)
{
    int i,j,k;
    int max_r;// 当前这列绝对值最大的行.
    int col;//当前处理的列
    int ta,tb;
    int LCM;
    int temp;
    int free_x_num;
    int free_index;

    for(int i=0;i<=var;i++)
    {
        x[i]=0;
        free_x[i]=true;
    }

    //转换为阶梯阵.
    col=0; // 当前处理的列
    for(k = 0;k < equ && col < var;k++,col++)
    {// 枚举当前处理的行.
// 找到该 col 列元素绝对值最大的那行与第 k 行交换.(为了在除法时减小误差)
        max_r=k;
        for(i=k+1;i<equ;i++)
        {
            if(abs(a[i][col])>abs(a[max_r][col])) max_r=i;
        }
        if(max_r!=k)
        {// 与第 k 行交换.
            for(j=k;j<var+1;j++) swap(a[k][j],a[max_r][j]);
        }
        if(a[k][col]==0)
        {// 说明该 col 列第 k 行以下全是 0 了，则处理当前行的下一列.
            k--;
            continue;
```

```
            }
            for(i=k+1;i<equ;i++)
            {// 枚举要删去的行.
                if(a[i][col]!=0)
                {
                    LCM = lcm(abs(a[i][col]),abs(a[k][col]));
                    ta = LCM/abs(a[i][col]);
                    tb = LCM/abs(a[k][col]);
                    if(a[i][col]*a[k][col]<0)tb=-tb;//异号的情况是相加
                    for(j=col;j<var+1;j++)
                    {
                        a[i][j] = a[i][j]*ta-a[k][j]*tb;
                    }
                }
            }
        }


        // 1. 无解的情况: 化简的增广阵中存在(0, 0, ..., a)这样的行(a != 0).
        for (i = k; i < equ; i++)
        { // 对于无穷解来说，如果要判断哪些是自由变元，那么初等行变换中的交换就会影响，则要记录交换.
            if (a[i][col] != 0) return -1;
        }
        // 2. 无穷解的情况: 在 var * (var + 1)的增广阵中出现(0, 0, ..., 0)这样的行，即说明没有形成严格的上三角阵.
        // 且出现的行数即为自由变元的个数.
        if (k < var)
        {
            // 首先，自由变元有 var - k 个，即不确定的变元至少有 var - k 个.
            for (i = k - 1; i >= 0; i--)
            {
                // 第 i 行一定不会是(0, 0, ..., 0)的情况，因为这样的行是在第 k 行到第 equ 行.
                // 同样，第 i 行一定不会是(0, 0, ..., a), a != 0 的情况，这样的无解的.
                free_x_num = 0; // 用于判断该行中的不确定的变元的个数，如果超过 1 个，则无法求解，它们仍然为不确定的变元.也就是除了本身（因为到这一行为止，要求的变元也是处在不确定状态）以外的不确定的变元系数全为 0 的时候，这个变元也是唯一的
                for (j = 0; j < var; j++)
                {
                    if (a[i][j] != 0 && free_x[j]) free_x_num++, free_index = j;
                }
                if (free_x_num > 1) continue; // 无法求解出确定的变元.
                // 说明就只有一个不确定的变元 free_index，那么可以求解出该变元，且该变元是确定的.
                temp = a[i][var];
                for (j = 0; j < var; j++)
                {
                    if (a[i][j] != 0 && j != free_index) temp -= a[i][j] * x[j];
                }
                x[free_index] = temp / a[i][free_index]; // 求出该变元.
                free_x[free_index] = 0; // 该变元是确定的.
            }
            return var - k; // 自由变元有 var - k 个.
        }
        // 3. 唯一解的情况: 在 var * (var + 1)的增广阵中形成严格的上三角阵.
        // 计算出 Xn-1, Xn-2 ... X0.
        for (i = var - 1; i >= 0; i--)
        {
            temp = a[i][var];
            for (j = i + 1; j < var; j++)
            {
```

```
            if (a[i][j] != 0) temp -= a[i][j] * x[j];
        }
        if (temp % a[i][i] != 0) return -2; // 说明有浮点数解，但无整数解.在这里如果要求解也是可以的
        x[i] = temp / a[i][i];
    }
    return 0;
}
```

6.10.4 经典题目

6.10.4.1 题目 1

1．题目出处/来源

POJ-2065SETI

2．题目描述•

太空和地面之间要传递一些信息，并且找到了一种对信息处理的多项式的方法，对于每一个小写字母，先采用数字映射的方法，即 1 代表 a,2 代表 b……26 代表 z 而 0 代表*，传递信息的时候用一组特定的数字 a0——an-1 对于每一个映射的数字用一个多项式的函数 f(k)=∑ai*ki(mod p)(0<=i<n-1,1<=k<=n，n 表示要传递的信息的长度，f(k)表示第 k 个字母表示的映射)来表示，p 为素数，而且 p>n 且 p>26，要求你求出对于特定的信息的数字序列的组合。

3．分析

这道题给出了线性方程组的一般形式，即 f(k)=a0*k0+a1*k1+….+an-1*kn-1(mod P)我们只要按照给出的形式列出线性方程组进行取模求解就可以了

4．代码（包含必要注释，采用最适宜阅读的 Courier New 字体，小五号，间距为固定值 12 磅）

```
#include<iostream>
#include<cstdio>
#include<cstring>
#define MAXN 75
using namespace std;
int Abs(int a)
{
    return a<0?-a:a;
}
int a[MAXN][MAXN];//增广矩阵
int x[MAXN];//解集
bool free_x[MAXN];//标记是否是不确定的变元
int mod;
int gcd(int a,int b)
{
    int t;
    while(b!=0)
    {
        t=b;
        b=a%b;
        a=t;
    }
    return a;
}
int lcm(int a,int b)
{
    return a/gcd(a,b)*b;//先除后乘防溢出
}
```

```
// 高斯消元法解方程组(Gauss-Jordan elimination).(-2 表示有浮点数解，但无整数解，
//-1 表示无解，0 表示唯一解，大于 0 表示无穷解，并返回自由变元的个数)
//有 equ 个方程，var 个变元。增广矩阵行数为 equ,分别为 0 到 equ-1,列数为 var+1,分别为 0 到 var.
int Gauss(int equ,int var)
{
    int i,j,k;
    int max_r;// 当前这列绝对值最大的行.
    int col;//当前处理的列
    int ta,tb;
    int LCM;
    int temp;
    int free_x_num;
    int free_index;

    for(int i=0;i<=var;i++)
    {
        x[i]=0;
        free_x[i]=true;
    }

    //转换为阶梯阵.
    col=0; // 当前处理的列
    for(k = 0;k < equ && col < var;k++,col++)
    {// 枚举当前处理的行.
// 找到该 col 列元素绝对值最大的那行与第 k 行交换.(为了在除法时减小误差)
        max_r=k;
        for(i=k+1;i<equ;i++)
        {
            if(Abs(a[i][col])>Abs(a[max_r][col])) max_r=i;
        }
        if(max_r!=k)
        {// 与第 k 行交换.
            for(j=k;j<var+1;j++) swap(a[k][j],a[max_r][j]);
        }
        if(a[k][col]==0)
        {// 说明该 col 列第 k 行以下全是 0 了，则处理当前行的下一列.
            k--;
            continue;
        }
        for(i=k+1;i<equ;i++)
        {// 枚举要删去的行.
            if(a[i][col]!=0)
            {
                LCM = lcm(Abs(a[i][col]),Abs(a[k][col]));
                ta = LCM/Abs(a[i][col]);
                tb = LCM/Abs(a[k][col]);
                if(a[i][col]*a[k][col]<0)tb=-tb;//异号的情况是相加
                for(j=col;j<var+1;j++)
                {
                    a[i][j] = ((a[i][j]*ta)%mod-(a[k][j]*tb)%mod+mod)%mod;
                }
            }
        }
    }
    // 3. 唯一解的情况: 在 var * (var + 1)的增广阵中形成严格的上三角阵.
    // 计算出 Xn-1, Xn-2 ... X0.
    for (i = var - 1; i >= 0; i--)
    {
        temp = a[i][var];
```

```
        for (j = i + 1; j < var; j++)
        {
            if (a[i][j] != 0) temp -= a[i][j] * x[j];
            temp=(temp%mod+mod)%mod;
        }
        while(temp%a[i][i])
        temp+=mod;
        x[i] = (temp / a[i][i])%mod;
    }
    return 0;
}
int Q_mod(int a,int b,int m)//快速模幂运算
{
    int res=1;
    a%=m;
    while(b)
    {
        if(b&1)
        {
            res*=a;
            res%=m;
        }
        a*=a;
        a%=m;
        b>>=1;
    }
    return res;
}
int main()
{
    int T,i,j,k,len;
    scanf("%d",&T);
    char s[MAXN];
    while(T--)
    {
        scanf("%d%s",&mod,s);
        len=strlen(s);
        memset(a,0,sizeof(a));
        for(i=0;i<len;i++)
        {
            if(s[i]=='*')
            a[i][len]=0;
            else
            a[i][len]=s[i]-'a'+1;
            for(j=0;j<len;j++)
            a[i][j]=Q_mod(i+1,j,mod);
        }
        Gauss(len,len);//题目保证有解
        for(i=0;i<len;i++)
        {
            if(i)
            printf(" ");
            printf("%d",x[i]);
        }
        printf("\n");
    }
    return 0;
}
```

5. 思考与扩展：可以借鉴北大培训教材中做法。

6.10.4.2 题目 2

1．题目出处/来源

POJ-1222 EXTENDED LIGHTS OUT

2．题目描述

在一个 5*6 的开关矩阵里，有的开关是开的状态，有的是关的状态，每一个开关的状态变化的时候都会引起其上下左右四个开关的状态一起变化，因为每个开关进行两次变化和不变化是一样的，所以每个开关的状态对多变化一次，现在给出来这个开关矩阵的起始状态（1 表示开，0 表示关），问题是怎么样能让所有的开关都变为关闭状态，输出操作矩阵，1 表示这个位置的开关状态要变化，0 表示不变化

3．分析

对于每一个开关的状态都由自己和周围的几个开关共同决定，而且每个开关只能变化一次，所以我们可以对每一个开关都建立一个线性方程，因为这里面只有 0 和 1 所以最后的运算就变成了模 2 的运算，用异或就会方便很多。

4．代码（包含必要注释，采用最适宜阅读的 Courier New 字体，小五号，间距为固定值 12 磅）

```
#include<iostream>
#include<cstring>
#include<cstdio>
#define N 30
using namespace std;
int a[N][N];
int x[N];//存储每一个位置对应的最终状态，同时存储每一个未知数的解
int n;
int Abs(int a)
{
   return a<0?-a:a;
}
void gauss()
{
   int i,j,k,row;
   n=30;
   for(k=0;k<n;k++)//一次缩减矩阵的阶数
   {
      for(i=k;i<n;i++)//对每一个未知数找到第一个本列是 1 的方程
      if(a[i][k]==1)
      {
         row=i;
         break;
      }
      if(row!=k)//交换原来的列和第一个本列是 1 的方程，构成行阶梯型原始状态
      {
          for(j=k;j<n;j++)
        {
            swap(a[i][j],a[k][j]);
         }
         swap(x[i],x[k]);
      }
      for(i=k+1;i<n;i++)//行阶梯型的化简
      {
         if(a[i][k]==0)
         continue;
         for(j=k+1;j<n;j++)
         a[i][j]^=a[k][j];
         x[i]^=x[k];
      }
```

```
        }
        x[n-1]/=a[n-1][n-1];
        for(i=n-2;i>=0;i--)//求解过程
        {
            for(j=i+1;j<n;j++)
            x[i]^=(x[j]*a[i][j]);//用异或运算简化运算过程
            x[i]/=a[i][i];
        }
    }
    void init()//方程未知数的系数
    {
        int i,j,t1,t2,t3,t4;
        for(i=0;i<30;i++)
        {
            t1=i/6;
            t2=i%6;
            for(j=0;j<30;j++)
            {
                t3=j/6;
                t4=j%6;
                if(Abs(t3-t1)+Abs(t4-t2)<=1)
                a[i][j]=1;
                else
                a[i][j]=0;
            }
        }
    }
    int main()
    {
        int t,i,j,k;
        scanf("%d",&t);
        for(k=1;k<=t;k++)
        {
            for(i=0;i<30;i++)
            scanf("%d",&x[i]);
            init();
            gauss();
            printf("PUZZLE #%d\n",k);
            for(i=0;i<5;i++)
            {
                for(j=0;j<6;j++)
                {
                    if(j)
                    printf(" ");
                    printf("%d",x[i*6+j]);
                }
                printf("\n");
            }
        }
        return 0;
    }
```

6.10.5 扩展变型

[POJ-2947Widget Factory]

第7章 组合数学

7.1 母函数

参考文献：
ACM-ICPC 程序设计系列 组合数学及其应用
扩展阅读：

编写：陈禹　　校核：曹振海

7.1.1 基本原理

在数学中，某个序列 的母函数是一种形式幂级数，其每一项的系数可以提供关于这个序列的信息。使用母函数解决问题的方法称为母函数方法。

母函数可分为很多种，包括普通母函数、指数母函数、L 级数、贝尔级数和狄利克雷级数。对每个序列都可以写出以上每个类型的一个母函数。构造母函数的目的一般是为了解决某个特定的问题，因此选用何种母函数视乎序列本身的特性和问题的类型。

母函数的表示一般使用解析形式，即写成关于某个形式变量 x 的形式幂级数。对幂级数的收敛半径中的某一点，可以求母函数在这一点的级数和。但无论如何，由于母函数是形式幂级数的一种，其级数和不一定对每个 x 的值都存在。

母函数方法不仅在概率论的计算中有重要地位，而且已成为组合数学中一种重要方法。此外，母函数在有限差分计算、特殊函数论等数学领域中都有着广泛的应用。

注意母函数本身并不是一个从某个定义域射到某个上域的函数，名字中的“函数”只是出于历史原因而保留。

对于数列，称无穷级数为该数列的（普通型）母函数，简称普母函数或母函数。

例如有限数列 C(n,r)，r＝0,1,2, …,n 的母函数是 $(1+x)^n$ 。

定理一：　组合的母函数：设 $S=\{n_1\cdot e_1,n_2\cdot e_2,\cdots,n_m\cdot e_m\}$,且 n1＋ n2＋…＋ nm＝n，则 S 的 r 可重组合的母函数为

$$G(x)=\prod_{i=1}^{m}\left(\sum_{j=0}^{n_i}x^j\right)=\sum_{r=0}^{n}a_r x^r$$

其中，r 可重组合数为 x^r 之系数 a_r ，r＝0,1,2, …,n .

7.1.2 解题思路

通过题意来构造母函数

有质量为 1,2,3,的砝码各一枚，问：

可以称出多少种不同质量的物品？

假设用 x 表示砝码，x 的示数表示质量可以得到：

1 个 1g 砝码可以用多项式 1+x 表示，这里 1 表示可以不用质量为 1 的砝码，x 表示可以用质量为 1 的砝码，质量为 1。

1 个 1g 砝码可以用多项式 1+x^2 表示，这里 1 表示可以不用质量为 2 的砝码，x^2 表示可以用质量为 2 的砝码，质量为 2。

1 个 1g 砝码可以用多项式 1+x^3 表示，这里 1 表示可以不用质量为 3 的砝码，x^3 表示可以用质量为 3 的砝码，质量为 3。

根据定理 1，构造母函数如下

G(x) = (1 + x)(1 +x)(1+x^2)=1 +x + x^2+2x^3+x^4+x^5+x^6

从上面可以知道，可以称出质量为 1~6,6 种质量，而系数就是每种质量的方案数。

x^3 的系数是 2，因此要称出质量为 3 的物品，有 2 种可能的方案。

假设上面的每种砝码有无数多个，这种情况对应的母函数是什么?

以 2g 砝码为例，可以用多项式 1+x+x^4+x^6+…..来表示，这里的 1 仍然用来表示不用 2g 砝码，x^2 表示使用一个 2g 砝码，x^4 表示使用俩个 2g 砝码，x^6 表示使用三个 2g 砝码，以此类推下去。

因此构造的母函数如下

G(x) = (1 + x+x^2+……)(1 +x^2+x^4+…..)(1+x^3+x^6+……)

通过以上例子只要基本理解母函数构造方法即可!!!

7.1.3 模板代码

7.1.4 经典题目

1．题目出处/来源

hdu2082 找单词

2．题目描述

假设有 x1 个字母 A， x2 个字母 B,..... x26 个字母 Z，同时假设字母 A 的价值为 1，字母 B 的价值为 2,..... 字母 Z 的价值为 26。那么，对于给定的字母，可以找到多少价值<=50 的单词呢？单词的价值就是组成一个单词的所有字母的价值之和，比如，单词 ACM 的价值是 1+3+14=18，单词 HDU 的价值是 8+4+21=33。(组成的单词与排列顺序无关，比如 ACM 与 CMA 认为是同一个单词)。

3．分析

这是一个标准的母函数题目，所要求解的问题可以转化为求 x 的指数小于等于 50 的系数和

4．代码（包含必要注释，采用最适宜阅读的 Courier New 字体，小五号，间距为固定值 12 磅）

```
#include<stdio.h>
int main()
{
   int n,i,j,k;
   scanf("%d",&n);
   while(n--) {
      int a[27] = {0}, b[100]={0}, c[100] = {0};
      for(i = 1; i <= 26; ++i)//输入每个字母的价值
      scanf("%d", &a[i]);
      for(i = 1; i <= 26; i++) {//先将第一件物品小于等于 50 指数的系数初始化为 1
         if(a[i] != 0) {
            for(j = 0; i * j <= 50 && j <= a[i]; j++)
               b[i*j]=1;
            break;
         }
      }
      for(i = i + 1; i <= 26; i++) {对于剩下的物品求出指数小于等于 50 的系数
        if(a[i] != 0) {
           for(k = 0; k <= 50; k++) {
              if(b[k] != 0) {
                  for(j = 0; j * i + k <= 50 && j <= a[i]; j++) {
```

```
                        c[i * j + k] += b[k];
                    }
                }
            }
            for(k = 0; k <= 50; k++) {
              b[k] = c[k];
              c[k] = 0;
            }
          }
        }
        int sum = 0;
        for(i = 1; i <= 50; i++) {
           sum += b[i];
        }
        printf("%d\n", sum);
    }
    return 0;
}
```

7.1.5 扩展变型

7.2 置换群

参考文献：

ACM-ICPC 程序设计系列 组合数学及其应用

编写：陈禹　　校核：曹振海

7.2.1 基本原理

定义 1　任一集合 A 到自身的映射都叫做 A 的一个变换，如果 A 是有限集且变换是一一变换（双射），

那么这个变换为 A 的一个置换。有限集合 A 的若干个置换若作成群，就叫做置换群。含有 n 个元素的有限群 A 的全体置换作成的群，叫做 n 次对称群。通常记为 S_n.

说明：由定义 1 知道，置换群就是一种特殊的变换群（即有限集合上的变换群）而 n 次对称群 S_n 也就是有限集合 A 的完全变换群。

现以 $A=\{a_1,a_2,a_3\}$ 为例，设 π：$A\to A$ 是 A 的一一变换。即 π：$a_1\mapsto a_2, a_2\mapsto a_3, a_3\mapsto a_1$,利用本教材中特定的表示方法有：$a_1^{\pi}=a_2$，$a_2^{\pi}=a_3$，$a_3^{\pi}=a_1$.由于映射中只关心元素之间的对称关系.而不在乎元素的具体内容.故可设 $A=\{1,2,3\}$.故此.π:$1\mapsto 2, 2\mapsto 3, 3\mapsto 1$.稍做修改：

$$\pi:\begin{matrix}1&2&3\\ \downarrow&\downarrow&\downarrow\\ 2&3&1\end{matrix}\quad\Rightarrow\quad \pi=\begin{pmatrix}1&2&3\\2&3&1\end{pmatrix}.$$

用 $\pi=\begin{pmatrix}1&2&3\\2&3&1\end{pmatrix}$ 来描述 A 的一个置换的方便之处是显而易见的.当然,上述的置换可记为

$$\begin{pmatrix}2&1&3\\3&2&1\end{pmatrix},\ \begin{pmatrix}3&1&2\\1&2&3\end{pmatrix}\ldots,$$

但习惯上都将第一行按自然序列排写这就可以让我们都统一在一种表示置换的方法内

进行研究工作了.习惯上称它为三元置换.

二.置换的乘积.

设 $A=\{1,2,3\}$ 的任二个置换为 $\pi=\begin{pmatrix}1&2&3\\2&3&1\end{pmatrix}$, $\tau=\begin{pmatrix}1&2&3\\3&1&2\end{pmatrix}$ 那么由于 π 和 τ 都是一一变换,于是 $\pi\tau$ 也是 A 的一一变换.且有

$\pi\tau:1\to1,2\to2,3\to3$； $1^{\tau\pi}=1,2^{\tau\pi}=2,3^{\tau\pi}=3$.

$$\pi\tau=\begin{pmatrix}1&2&3\\2&3&1\end{pmatrix}\begin{pmatrix}1&2&3\\3&1&2\end{pmatrix}=\begin{pmatrix}1&2&3\\1&2&3\end{pmatrix}$$

换句话说:

注意:置换乘积中,是从左到右求变换值,这是与过去的习惯方法不同的。

例 2 设 $A=\{1,2,3\}$, 那么 A 的全部一一变换构成的三次对称群为 $S_3=\{\pi_0,\pi_1,\pi_2,\pi_3,\pi_4,\pi_5\}$.其中

$$\pi_0=\begin{pmatrix}1&2&3\\1&2&3\end{pmatrix},\quad \pi_1=\begin{pmatrix}1&2&3\\1&3&2\end{pmatrix},\quad \pi_2=\begin{pmatrix}1&2&3\\2&1&3\end{pmatrix}$$

$$\pi_3=\begin{pmatrix}1&2&3\\2&3&1\end{pmatrix},\quad \pi_4=\begin{pmatrix}1&2&3\\3&1&2\end{pmatrix},\quad \pi_5=\begin{pmatrix}1&2&3\\3&2&1\end{pmatrix}$$

所以 $|S_3|=3!=6$.其中 π_0 是恒等变换.即 π_0 是 S_3 的单位元.

循环置换(轮换)

前面我们已经引入了置换的记法,下面,再介绍一种记法.设有 8 元置换 $\pi=\begin{pmatrix}1&2&3&4&5&6&7&8\\4&3&5&2&1&6&7&8\end{pmatrix}$,

π 的变换过程为 $1\to4\to2\to3\to5\to1$,即其他元素都不改变,若将不发生改变的文字都删掉,那么上述置换可写成循环置换的形式: $\pi=(1\quad 4\quad 2\quad 3\quad 5)$

$\begin{pmatrix}1&2&3&4&5\\2&3&1&4&5\end{pmatrix}=(1\quad 2\quad 3)$ 叫作 3—循环置换.

$\begin{pmatrix}1&2&3&4&5\\2&3&4&5&1\end{pmatrix}=(1\quad 2\quad 3\quad 4\quad 5)$ 叫作 5—循环置换.

$\begin{pmatrix}1&2&3&4&5\\1&2&3&4&5\end{pmatrix}=(1)$ 叫作 1—循环置换.

7.2.2 解题思路

7.2.3 模板代码

7.2.4 经典题目

1．题目出处/来源

hrbust OJ 1536 Leonardo's Notebook

2．题目描述

给出一个 A~Z 的置换，问是否可以被表示为一个置换的平方（即 G=G'*G'）。

3．分析

通过观察可以发现，一个置换乘上它本身，其中长度为偶数的循环节必然会分裂为两个长度相等的循环节，长度为奇数的循环节还是一个循环节，长度不变。如：

2341*2341=3412 （2341 是一个循环节，平方后分裂为两个长度为 2 的循环节[13][24]）

23451*23451=34512 (23451 是一个循环节，平方后还是长度为 5 的循环节)

所以，给出的置换中长度为偶数的循环节必然是原置换中的循环节分裂出来的，而长度为奇数的循环节有可能是原置换中的循环节分裂出来的。因为只要判断能否被表示，所以可以忽略长度为奇数的循环节，只对长度为偶数的循环节进行考虑即可。

因为一个长度为偶数的循环节分裂出来的两个循环节的长度相等且同为原循环节长度的一半。所以，只要给出置换中所包含的长度为偶数的循环节能一一配对，那么就必然可以被表示为另一个置换的平方。

定理：任意一个长为 L 的置换的 k 次幂，会把自己分裂成 gcd(L,k) 分， 并且每一份的长度都为 L / gcd(l,k)

假如 d = gcd(L,K)，l = L / gcd(L,k)，那么我们只需要找到 d 个长为 l 的循环，将他们交错循环连接成一个长为 d * l 的大循环， 这样一个过程就相当于开 k 次方。

然后对每个长度为 len (1 <= len <= 26 ）度的循环都做一下判断，看能否找到 gcd (len, 2)个这样的循环，若找到了，那么说明这一部分可以开 2 次方。没找到就直接返回 false.

4．代码（包含必要注释，采用最适宜阅读的 Courier New 字体，小五号，间距为固定值 12 磅）

```
#include<stdio.h>
#include<string.h>
int main()
{
   char s[1000],s1[1000];
   int t, num[30], i, flag[30], len[30], now, l;
   scanf("%d",&t);
   while(t--) {
      memset(flag,0,sizeof(flag));
      memset(len,0,sizeof(len));
      scanf("%s",s);
      for(i=0;i<26;i++)
         num[i]=s[i]-'A';
      for(i=0;i<26;i++) {//求各种循环节长度
         if(flag[i])
            continue;
         now = i; l = 0;
         while(flag[now] == 0) {
            l++;
            flag[now] =1;
            now = num[now];
         }
         len[l]++;
      }
      for(i = 2; i <= 26; i += 2) {
         if(len[i] % 2 == 1)
            break;
      }
      if(i == 28)
         printf("Yes\n");
      else
```

```
            printf("No\n");
        }
        return 0;
    }
```

7.2.5 扩展变型

7.3 排列组合

参考文献：
《ACM-ICPC 程序设计系列<组合数学及应用>》
排列组合——百度百科

扩展阅读：
编写：曹振海　　校核：陈禹

7.3.1 基本原理

排列组合的相关知识高中已经有了深入的讲解，全排列和组合数等概念也都比较熟悉，排列组合的中心问题是研究给定要求的排列和组合可能出现的情况总数。排列组合有两个基本原理：

加法原理和分类计数法

加法原理：做一件事，完成它可以有 n 类办法，在第一类办法中有 m1 种不同的方法，在第二类办法中有 m2 种不同的方法，……，在第 n 类办法中有 mn 种不同的方法，那么完成这件事共有 N=m1+m2+m3+…+mn 种不同方法。

第一类办法的方法属于集合 A1，第二类办法的方法属于集合 A2，……，第 n 类办法的方法属于集合 An，那么完成这件事的方法属于集合 A1UA2U…UAn。

分类的要求 ：每一类中的每一种方法都可以独立地完成此任务；两类不同办法中的具体方法，互不相同（即分类不重）；完成此任务的任何一种方法，都属于某一类（即分类不漏）。

(2)乘法原理和分步计数法

乘法原理：做一件事，完成它需要分成 n 个步骤，做第一步有 m1 种不同的方法，做第二步有 m2 种不同的方法，……，做第 n 步有 mn 种不同的方法，那么完成这件事共有 N=m1×m2×m3×…×mn 种不同的方法。

合理分步的要求,任何一步的一种方法都不能完成此任务，必须且只须连续完成这 n 步才能完成此任务；各步计数相互独立；只要有一步中所采取的方法不同，则对应的完成此事的方法也不同。

排列组合还包括几个重要的公式：

排列的定义及其计算公式：从 n 个不同元素中，任取 m(m≤n,m 与 n 均为自然数,下同）个元素按照一定的顺序排成一列，叫做从 n 个不同元素中取出 m 个元素的一个排列；从 n 个不同元素中取出 m(m≤n）个元素的所有排列的个数，叫做从 n 个不同元素中取出 m 个元素的排列数，用符号 A(n,m）表示。A(n,m)=n(n-1)(n-2)……(n-m+1)= n!/(n-m)! 此外规定 0!=1

组合的定义及其计算公式：从 n 个不同元素中，任取 m(m≤n）个元素并成一组，叫

做从 n 个不同元素中取出 m 个元素的一个组合；从 n 个不同元素中取出 m(m≤n）个元素的所有组合的个数，叫做从 n 个不同元素中取出 m 个元素的组合数。用符号 C(n,m) 表示。C(n,m)==A(n,m)/m！；C(n,m)=C(n,n-m)。（n>=m)

其他排列与组合公式 从 n 个元素中取出 m 个元素的循环排列数=A(n,m)/m=n!/m(n-m)!. n 个元素被分成 k 类，每类的个数分别是 n1,n2,...nk 这 n 个元素的全排列数为 n!/(n1！×n2！×...×nk!). k 类元素，每类的个数无限，从中取出 m 个元素的组合数为 C(m+k-1,m)。

7.3.2 解题思路

排列组合的题目比较少，在计算排列组合的时候一定要注意乘法越界的问题，然后就是公式的应用，组合通常也会和状态 DP 以及深度优先搜索联系起来，其实状态 DP 的每个状态以及深度优先搜索所得到的每一个解都可以当成是一个组合。

7.3.3 模板代码

7.3.4 经典题目

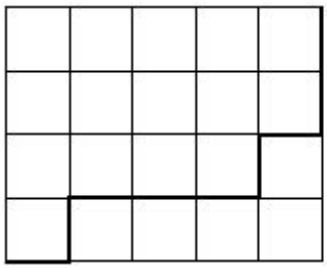
7.3.4.1 题目 1

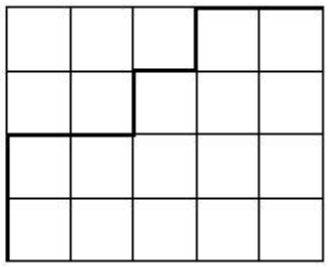
1．题目出处/来源

HLG1123-Grid

2．题目描述

给出下图的一个 n*m 的一个 grid，从左下角出发到达目的地右上角,每次只能沿着图中的线来走，而且每次只能向上或者向右行走，如下图所示两种不同的走法





你能求出从起始点到达目的地的所有不同种类的路径数目吗

3．分析

这道题有两种解法，观察图形我们可以发现，每一个格子都只能由下方和左方的点到达，所以可以利用递推得出到达每个点的不同走法的种类数。第二种方法是利用组合数，我们必须走 m 个向上的和 n 个向右的，那么总共的走法的数量就相当于从 m+n 个数中选出 n 个数的组合的数量

4．代码（包含必要注释，采用最适宜阅读的 Courier New 字体，小五号，间距为固定值 12 磅）

```
#include<stdio.h>
int main()
{
int m,n,i,sum1,sum2;
while(scanf("%d%d",&m,&n),(m||n))
{
```

```
        sum1=1;sum2=1;
        for(i=n+1;i<=m+n;i++)
            sum1*=i;
        for(i=1;i<=m;i++)
            sum2*=i;
        printf("%d\n",sum1/sum2);
    }
    return 0;
    }
```

5．思考与扩展：可以借鉴北大培训教材中做法。

7.3.4.2 题目 2

1．题目出处/来源

POJ2249-Binomial Showdown

2．题目描述

题意即为从 n 个数种选出 k 个有多少种情况

3．分析

这道题直接利用组合数的公式，但是在计算过程中有可能越界，所以最好先把组合数公式中分子的每一个数和分母的每一个数的最大公约数除掉，把分子全部变为 1，这时只要把分子剩下的连乘就可以了。

4．代码（包含必要注释，采用最适宜阅读的 Courier New 字体，小五号，间距为固定值 12 磅）

```
#include <iostream>
#include <cstdio>
#include <cstring>
#define N 2000
using namespace std;
int p[N],q[N];
int gcd(int a,int b)
{
if(!a)
    return b;
int c;
while(b)
{
    c = b;
    b = a%b;
    a = c;
}
return a;
}
int main()
{
int n,m,i,j,k;
while(scanf("%d%d",&n,&m)&&(n || m))
{
    if(m > n - m)
        m = n - m;
    int temp=n;
    for(i = 1;i <= m;i++)
    {
        p[i] = temp--;
        q[i] = i;
```

```
        }
        for(i = 1;i <= m;i++)
        {
            for(j = 1; j <= m;j++)
            {
                temp = gcd(p[i],q[j]);
                if(temp != 1)
                {
                    p[i] /= temp;
                    q[j] /= temp;//因为组合数是个整数，所以最后一定可以把分母化为 1
                    if(p[i] == 1)
                        break;
                }
            }
        }
        int sum = 1;
        for(i = 1;i <= m;i++)
            sum *= p[i];
        printf("%d\n",sum);
    }
    return 0;
    }
```

7.4 卡塔兰数(Catalan)

参考文献：
卡特兰数-百度百科

扩展阅读：
卡特兰数更多介绍和讲解：
http://blog.csdn.net/ffjjqqjj/article/details/6081711

编写：曹振海　　校核：陈禹

7.4.1 基本原理

卡特兰数可以看做是一个与 n 有关的数列，并且有自己的递推公式以及通项公式，其一个递推公式为 h(n)=h(0)*h(n-1)+h(1)*h(n-2)+..h(n-1)*h(0)另一个递推公式为 h(n)=(h(n-1)*(4*n-2))/(n+1),这个式子的解为 h(n)=C(2n,n)/(n+1)，单独看公式是没有意义的，卡特兰数的应用有很多方面，比如有 n 个左括号,n 个又括号，可以组成的合法的括号序列有多少个，这个数的解就是卡特兰数，还有 2n 个人买票的问题，有 n 个人有 5 元钱，n 个人有 10 元钱，票价为 5 元，求共有多少种排队方式可以保证对每一个拿十元的人都有 5 元钱能找给他，这个解也是卡特兰数。另外还有 n 个数依次入栈，每个数只能入栈出栈一次，有多少种出栈序列，etc

7.4.2 解题思路

一般情况不会有直接求卡特兰数的题，需要一定的抽象过程和推理工程，不过我们在遇到和栈、括号、二叉树的种类数（用第一个递推公式比较容易理解）等有关的题目时，不妨试一下卡特兰数的思路，更多情况下我们都能把题目给定的模型抽象成为一个合法括号序列的模型，比如说买票的问题和出栈序列的问题都可以归结到合法括号序列的问题上来。

7.4.3 模板代码

直接套用公式或利用题目中给出的递推关系

7.4.4 经典题目

7.4.4.1 题目 1

1．题目出处/来源

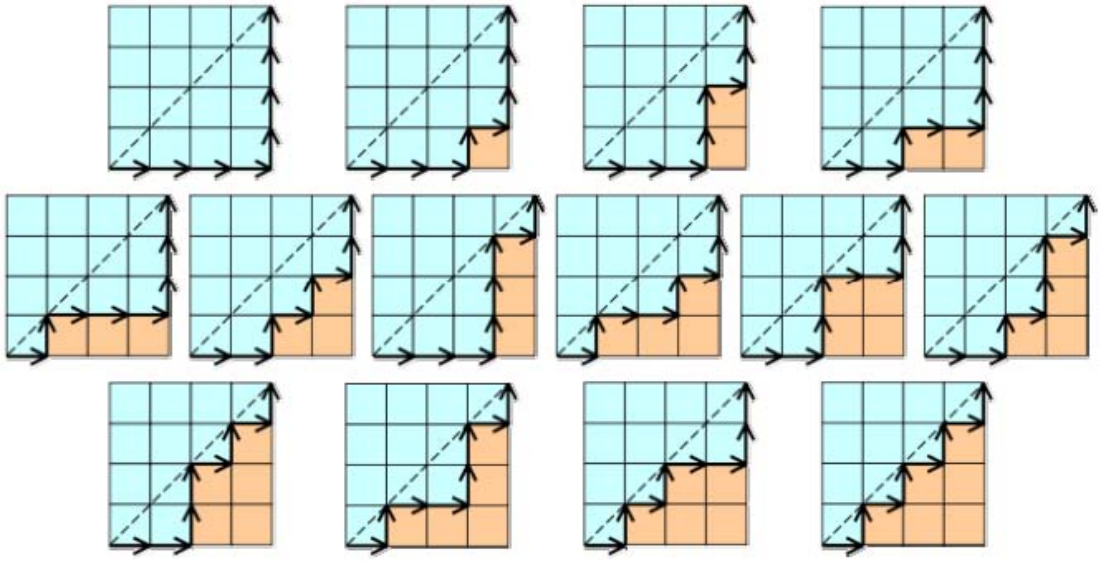
HDU2067-小兔的棋盘

2．题目描述

小兔的叔叔从外面旅游回来给她带来了一个礼物，小兔高兴地跑回自己的房间，拆开一看是一个棋盘，小兔有所失望。不过没过几天发现了棋盘的好玩之处。从起点(0，0)走到终点(n,n)的最短路径数是 C(2n,n),现在小兔又想如果不穿越对角线(但可接触对角线上的格点)，这样的路径数有多少?小兔想了很长时间都没想出来，现在想请你帮助小兔解决这个问题，对于你来说应该不难吧!

3．分析

这道题的答案是 n 对应的卡特兰数的二倍，我们最关心的是如何归结到卡特兰数的计算上来，首先因为图关于对角线是对称的，所以我们只看一半就可以了，要到达点(n,n)就要保证往右走和往上走的步数都为 n，另外，为了保证不能越过对角线，必须保证在任何一步往右走的步数不小于往上走的步数，这不就是合法的括号序列问题了吗？那么我们就可以直接利用递推公式求解了。



4．代码（包含必要注释，采用最适宜阅读的 Courier New 字体，小五号，间距为固定值 12 磅）

```
#include <iostream>
#include <cstdio>
#include <cstring>
#define N 40
using namespace std;
_int64 cal[N];
int main()
{
_int64 i,j,n,t;
memset(cal,0,sizeof(cal));
cal[0]=cal[1]=1;
for(i=2;i<=35;i++)
```

```
        for(j=0;j<i;j++)
            cal[i]+=cal[j]*cal[i-j-1];//递推公式
    t=0;
    while(scanf("%I64d",&n)&&(~n))
    {
        printf("%I64d %I64d %I64d\n",++t,n,cal[n]*2);
    }
    return 0;
    }
```

7.5 容斥原理

参考文献：

《ACM-ICPC 程序设计系列<组合数学及应用>》

扩展阅读：

编写：曹振海　　校核：陈禹

7.5.1 基本原理

设 A1,A2 为有限集合，其元素个数分别为|A1|，|A2|,则| A1∪A2|=| A1+A2|-| A1∩A2|

这个定理，常称作包含排斥原理，也就是容斥原理。

容斥原理推广到有n个集合的情况，其元素个数分别为|A1|,|A2|,…，|An|,则| A1∪A2∪…∪An|=$\sum_{i=1}^{n} Ai$ - $\sum_{1<=i<j<=n} |Ai \cap Aj|$ + $\sum_{1<=i<j<k<=n} |Ai \cap Aj \cap Ak|$ +…+(-1)^n-1| A1∩A2∩…∩An|

7.5.2 解题思路

对于需要用到容斥原理的题型，一般都比较容易看出来用的方法，而且一般采用深搜的方法进行运算。

7.5.3 模板代码

7.5.4 经典题目

7.5.4.1 题目 1

1．题目出处/来源

2012 年第 37 届 ACM-ICPC 亚洲区预选赛金华站 J 题—HDU4451

2．题目描述

这道题题意是，有一些衣服和裤子和鞋，有些衣服和裤子以及裤子和鞋不能互相搭配，求出这些衣服裤子和鞋能搭配出多少套不同的服装。

3．分析

这道题属于比较简单的容斥定理，因为保证了不会出现衣服和鞋不能搭配的方式，所以只需要记录对一种裤子有多少个鞋不能搭配，有多少个裤子不能搭配，求出所有的搭配方式，然后减去不能搭配的方式，再加上因为同一个裤子产生的多减的情况，最后得出的就是所要求的解。

4．代码（包含必要注释，采用最适宜阅读的 Courier New 字体，小五号，间距为固定值 12 磅）

```
#include<iostream>
#include<cstring>
#include<cstdio>
#define N 1005
using namespace std;
int num[N][2];//num 数组记录与裤子不能搭配的衣服和鞋的数量
int main()
{
int n,m,k,i,j,t;
int sum;
int a,b;
char s1[10],s2[10];
while(scanf("%d%d%d",&n,&m,&k)&&(n||m||k))
{
    sum=n*m*k;
    scanf("%d",&t);
    memset(num,0,sizeof(num));
    while(t--)
    {
        scanf("%s%d%s%d",s1,&a,s2,&b);
        if(s1[0]=='c')
            num[b][0]++;
        else
            num[a][1]++;
    }
    for(i=1;i<=m;i++)
    {
        sum-=num[i][0]*k;
        sum-=num[i][1]*n;
        sum+=(num[i][0]*num[i][1]);//容斥定理
    }
    printf("%d\n",sum);
}
return 0;
}
```

5．思考与扩展：可以借鉴北大培训教材中做法。

7.5.4.2 题目 2

1．题目出处/来源

POJ2773-HAPPY

2．题目描述

给出一个数 m 和 k 要找出第 k 个和 m 互素的数

3．分析

这道题的容斥原理的方法可能不太好想，首先可以根据线性的性质来二分答案，并且判断 1-ans 之间有多少个数和 m 互素，互素的数的个数可以表示成为 ans-和 m 不互素的数的个数，所以可以找出在 ans 内有多少数和 m 不互素，这时候容斥定理就有了用武之地，根据 m 的范围可知，就算按照最小的质数 2 计算，其质因子数不会超过 20 个，所以可以在这 20 个质因子中利用容斥定理求出，这道题因为集合较多，所以要用深搜进行容斥定理的运算

4．代码（包含必要注释，采用最适宜阅读的 Courier New 字体，小五号，间距为固定值 12 磅）

```
#include<iostream>
#include<cstring>
#include<cstdio>
```

```
#define N 1000005
#define inf 0x7fffffff
using namespace std;
typedef long long LL;
bool isprime[N];
LL prime[N];
LL d[30],f;//这里开到 30 是因为即使按照最小的素数 2 来划分，也不会超过 20 个素因子，f 记录质因子数
int cnt;
void init()//素数表
{
   cnt=0;
   LL i,j;
   memset(isprime,true,sizeof(isprime));
   for(i=2;i<=N-5;i++)
   {
      if(isprime[i])
      {
         prime[cnt++]=i;
         for(j=i*i;j<=N-5;j+=i)
         isprime[j]=false;
      }
   }
}
void dfs(LL cur,LL now,LL mid,bool neg,LL &res)//容斥定理的搜索，cur 用来判断当前的层次 {
   if(cur>=f)
   return;
   LL n=now*d[cur];
   dfs(cur+1,now,mid,neg,res);//不乘的情况
   if(neg)
   res+=(mid/n);
   else
   res-=(mid/n);
   dfs(cur+1,n,mid,!neg,res);//乘的情况
}
LL sum(LL mid)//求 mid 内有多少和 m 互素的数
{
   LL res=0;
   dfs(0,1,mid,true,res);
   return mid-res;
}
int main()
{
   LL m,k,i,j,low,high,mid,ans,temp;
   init();
   while(scanf("%lld%lld",&m,&k)!=EOF)
   {
      f=0;
      for(i=0;i<cnt;i++)//m 的质因子分解
      {
         if(m%prime[i]==0)
         {
            d[f++]=prime[i];
            while(m%prime[i]==0)
            m/=prime[i];
         }
         if(m==1)
         break;
      }
      low=1,high=inf;
```

```
        while(low<=high)//二分答案
        {
            mid=(high-low)/2+low;
            temp=sum(mid);
            if(temp==k)
            ans=mid;
            if(temp>=k)
            high=mid-1;
            else
            low=mid+1;
        }
        printf("%lld\n",ans);
    }
    return 0;
}
```

扩展变型

7.6 Burnside 定理

参考文献：

《ACM-ICPC 程序设计系列<组合数学及应用>》

扩展阅读：

编写：陈禹　　校核：曹振海

7.6.1 基本原理

$$p = \underbrace{(a_1a_2\ldots a_{k_1})(b_1b_2\ldots b_{k_2})\ldots(h_1h_2\ldots h_{k_l})}_{l\text{ 项}}$$

$$k_1 + k_2 + \ldots + k_l = n$$

设其中 k 阶循环出现的次数为 ck，k=1,2,…,n,k 阶循环出现 ck 次，用(k)ck 表示。

Burnside 引理

设 G={a1,a2,…ag}是目标集[1,n]上的置换群。每个置换都写成不相交循环的乘积。G 将[1,n]换分成 n 个等价类。c1(ak)是在置换 ak 的作用下不动点的个数，也就是长度为 1 的循环的个数。

则有：n=[c1(a1)+c1(a2)+…+c1(ag)]/|G|

等价类

例：G={(1)(2)(3)(4),(12),(34),(12)(34)}.在 G 下，1 变 2，3 变 4，但 1 不变 3。

Z1=Z2={e,(3 4)}, Z3=Z4={e,(1 2)}.

对于 A4, Z1={e,(2 3 4),(2 4 3)},

Z2={e,(1 3 4),(1 4 3)} Z3={e,(1 2 4),(1 4 2)},

Z4={e,(1 2 3),(1 3 2)}

一般[1,n]上 G 将[1,n]分成若干等价类,满足等价类的 3 个条件.(a)自反性;(b)对称性;(c)传递性。一个由 G 定义的关系 k：

若存在 p∈G,使得 k→j 则称 kRj。显然 kRk；kRj 则 jRk；kRj,jRl 则 kRl。所以 R 是[1,n]上的一个等价关系。将[1,n]划分成若干等价类。

例：G={(1)(2)(3)(4),(12),(34),(12)(34)}.

1 和 2 属于一个等价类,3 和 4 属于一个等价类，即　E1=E2={1,2}，　E3=E4={3,4}

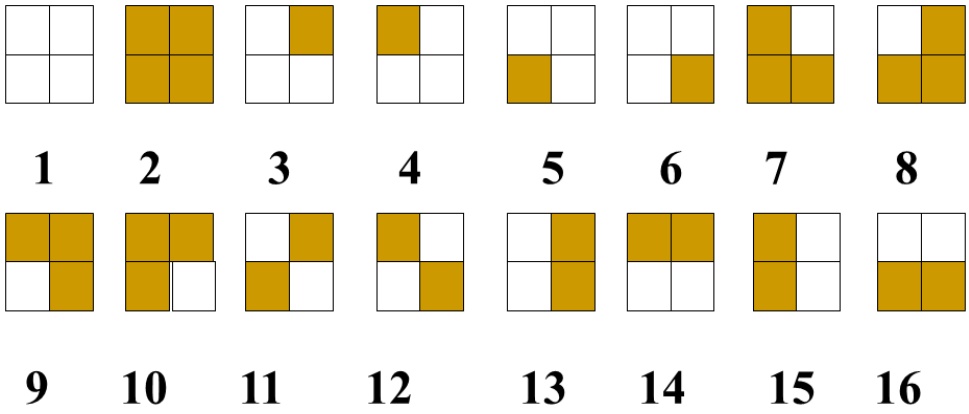
7.6.2 解题思路

例： G={e,(1 2),(3 4),(1 2)(3 4)}.

$$s_{jk}=\begin{cases}1\ ,若k\xrightarrow{a_j}k\\ 0\ ,若k\xrightarrow{a_j}l\quad(l\neq k)\end{cases}$$

s_{jk} 与 k / a_j	1 2 3 4	$c_1(a_j)$	格 式
(1)(2)(3)(4)	1 1 1 1	4	$(1)^4(2)^0(3)^0(4)^0$
(1 2)(3)(4)	0 0 1 1	2	$(1)^2(2)^1(3)^0(4)^0$
(1)(2)(3 4)	1 1 0 0	2	$(1)^2(2)^1(3)^0(4)^0$
(1 2)(3 4)	0 0 0 0	0	$(1)^0(2)^2(3)^0(4)^0$
$\|Z_k\|$	2 2 2 2	8	

例：一个正方形分成 4 格,2 着色,有多少种方案？图象：看上去不同的图形.方案：经过转动相同的图象算同一方案.图象数总是大于方案数。



1.旋转 0 度为不动置换：

p1=(1)(2)…(16) 不动点的个数为 16 个

2.逆时针转 90 度 ：

p2=(1)(2)(3 4 5 6)(7 8 9 10) (11 12)(13 14 15 16) 不动点的个数为 2 个

3.逆时针转 180 度 ：

p3=(1)(2)(3 5)(4 6)(7 9)(8 10) (11)(12)(13 15)(14 16) 不动点的个数为 4 个

4.逆时针转 270 度：

p4=(1)(2)(6 5 4 3)(10 9 8 7)(11 12)(16 15 14 13) 不动点的个数为 2 个

不同等价类的个数为(16+2+4+2)/4=6(种方案)

7.6.3 模板代码

n=[c1(a1)+c1(a2)+…+c1(ag)]/|G|

牢记此公式

G={a1,a2,…ag}是目标集[1,n]上的置换群，c1(ak)是在置换 ak 的作用下不动点的个数，也就是长度为 1 的循环的个数，n 代表有多少等价类。

7.6.4 经典题目

1．题目出处/来源

hrbust oj Smallbox 魔方

2．题目描述

n 阶魔方是一个 n×n×n 的立方体，有 6 个面，每个面又分为 n×n 个小块，每个小块都可以涂上不同的颜色，如右边的图中分别是 3 阶和 2 阶魔方。

高阶魔方同时兼备了收藏，鉴赏及实用价值，而高阶的魔方内部构造极为复杂，设计困难。针对这个问题，smallbox 公司推出了专门用于观赏的魔方，这种魔方不能拧动，为了能使魔方能展现不同的图案，smallbox 魔方配有 6 种不同颜色的贴纸，用户可以根据自己的喜好在每个小色块上贴上不同的颜色，纸一旦贴上就不能更改了，当然你仍然可以把整个魔方拿起来再换一个角度来摆放。

一个 smallbox 魔方配有六种不同颜色的贴纸，共 6×n×n 张。每种颜色的贴纸数量已知。现在由你来把这些贴纸贴到这个魔方上，使魔方表面的每个小色块都恰好被贴上一张贴纸，那么你有多少种不同的贴纸方案呢？

如果一种贴纸方案能通过变换摆放方式得到另一种方案，那么这两种方案算同一种。

3．分析

ai 代表每种颜色的数量

固定情况下的方案数

$$\frac{(6*n*n)!}{(a_1!*a_2!*a_3!*a_4!*a_5!*a_6!)}$$

模P环境下，除法使用乘以逆元实现

正方体的空间 24 个置换如下：

1. 绕中心轴转动(共 3*2 个) （n 分偶,奇讨论）

转动 270 度，90 度两种情况

若为偶数 有 长度为 4 的循环节。这循环节内的元素颜色要相同。才能保证是同一种方案。从而计算出方案总数。

若为奇数 有 长度为 4 的循环节和两个长度为 1 的循环节。

转动 180 度

若 n 为偶数每个循环节长度为 2

若 n 为奇数两个长度为 1 的循环节其他长度为 2。

2. 绕某个中心轴转 360 度（虽然有 3 个轴但是是同一个置换，共 1 个）

每个循环节长度为 1。

3. 绕连接对边的中心的轴转动 180 度（共 6 个）

循环节长度为 2 。(通过在魔方上标号，转动揣摩出来的)

4.绕体对角线轴转动 120 度,240 度 （共 4*2 个）

循环节长度为 3

所以 总共有 1+3*3+6*1+4*2=24 个置换。

奇偶共有：

4*2 个置换*循环节长度为 3 + 6 个置换*循环节长度为 2 + 1 个置换*循环节长度为 1

奇数还有:

(3*2)个置换*(两个长度为 1 的循环节其他长度为 4）+ 3 个置换*（两个长度为 1 的循环节其他长度为 4）

偶数还有:

(3*2)个置换*(循环节长度为 4)+ 3 个置换*（循环节长度为 2）

4．代码（包含必要注释，采用最适宜阅读的 Courier New 字体，小五号，间距为固定值 12 磅）

```
#include <iostream>
#include <cstdio>
#include <cstring>
using namespace std;
typedef long long LL;
const int max_n=15000+5;
const int MOD=1000000007;
LL   fac[max_n],inv[max_n],facv[max_n];      //fac[n]=n!%MOD   (facv[n]*fac[n])=1;
(inv[n]*n)%MOD=1
LL mypow(LL a,int b){
     LL res = 1;
     for(; b; b >>= 1){
        if(b & 1)  res = (res * a) % MOD;
        a *= a;a %= MOD;
     }
     return res;
}
LL cal_inv(int t){ //费马定理求逆元
     return mypow((LL)t, MOD - 2);
}
void ini() {//初始化逆元
     fac[0] = 1;   facv[0] = 1;
     for(int i = 1; i < max_n; i++) {
        inv[i] = cal_inv(i);
        facv[i] = (facv[i - 1] * inv[i]) % MOD;
        fac[i] = (fac[i - 1] * i) % MOD;
     }
}
int col[6], N;
int gcd(int a, int b){
     if(b == 0)  return a;
     return gcd(b, a % b);
}
LL solve(int t){//利用上面图中的排列组合公式求出方案数(会用到逆元)
     int b[6],tot = 0;
     for(int i = 0; i < 6; i++){
        b[i] = col[i] / t;
        tot += b[i];
     }
     LL res = 1;
     res = (res * fac[tot]) % MOD;
     for(int i = 0; i < 6; i++) {
```

```
            res = (res * facv[b[i]]) % MOD;
        }
        return res;
    }
    int main()
    {
        int T;
        ini();
        scanf("%d", &T);
        for(int ncas = 1; ncas <= T; ncas++){
            scanf("%d", &N);
            int e = 0;
            for(int i = 0; i < 6; i++){
                scanf("%d", &col[i]);
                e = gcd(e, col[i]);
            }
            LL ans = 0;
            ans = (ans + solve(1)) % MOD;
            if(e % 2 == 0) {
                ans = (ans + 6 * solve(2)) % MOD;
            }
            if(e % 3 == 0){
                ans = (ans + 8 * solve(3)) % MOD;
            }
            if(N & 1) {    //N 为奇数
                for(int i = 0; i < 6; i++){
                    for(int j = 0; j < 6; j++){
                        if((i == j && col[i] >= 2) || (i !=j && col[i] && col[j])){
                          col[i]--;   col[j]--;
                            int t = 0;
                            for(int k = 0;k < 6; k++){
                                t = gcd(t, col[k]);
                            }
                            if(t % 2 == 0){
                                ans = (ans + 3 * solve(2)) % MOD;
                            }
                          if(t % 4 == 0) {
                                ans = (ans + 6 * solve(4)) % MOD;
                            }
                            col[i]++;   col[j]++;
                        }
                    }
                }
            }
            else {
                if(e % 2 == 0) {
                   ans = (ans +3 * solve(2)) % MOD;
                }
                if(e % 4 == 0) {
                   ans = (ans + 6 * solve(4))%MOD;
                }
            }
            ans = (ans * inv[24]) % MOD;
            printf("%lld\n", ans, N);
        }
        return 0;
    }
```

7.7 Polya 计数

参考文献：

《ACM-ICPC 程序设计系列<组合数学及应用>》

扩展阅读：

编写：陈禹　　校核：曹振海

7.7.1 基本原理

Polya 定理：设 G={p1,p2,…,pg}是Ω上的一个置换群，C(pk)

是置换 pk 的循环的个数，用 M 中的颜色对Ω中的元素着色，着色方案数为：1/|G|*[m^c(p1)+m^c(p2)+m^c(p3)+...m^c(p[g])]|G|为置换的总个数,m 颜色数。c(pi)指的是置换 pi 的循环个数。

7.7.2 解题思路

我们用 Polya 定理来计算上节的用俩种颜色给四个方格着色问题，在 2*2 的方格中共有四个格子需要着色，分别标以 1,2,3,4，题目转化为用俩种颜色对 1,2,3,4 这四个面涂色。有四种置换方式即旋转方式。

顺时针旋转 0

p1=(1)(2)(3)(4)

顺时针旋转 90

p2=(1432)

顺时针旋转 180

p3=(13)(24)

顺时针旋转 270

p4=(1234)

c(p1)=4,c(p2)=1,c(p3)=2,c(p4)=1

根据 Polya 定理不同的着色方案为：

1/|G|*[m^c(p1)+m^c(p2)+m^c(p3)+...m^c(p[g])] = 1/4(2^4+2^1+2^2+2^1) =6。

这与我们用 Bunside 定理计算出来的结果一样。

7.7.3 模板代码

7.7.4 经典题目

7.7.4.1 题目 1

1．题目出处/来源

poj2409 Let it Bead

2．题目描述

给定颜色种数和环上的珠子总数，问有多少种染色方案（通过旋转和翻转相同算同一种）最后的结果不会超过 int 类型数据表示的范围。(俩条项链相同，当且仅当俩条项链通过旋转或者翻转能重合在一起，且对应珠子的颜色相同)

3．分析

(1)我们现在来讨论下通过旋转的情况，将项链顺时针旋转 i 格后，其循环节为 gcd(n,i) 即 n 与 i 最大公约数;

(2)通过旋转的情况:

当 n 为奇数时，共有 n 个循环节为(n+1)/2 的循环群。

当 n 为偶数时，共有 n/2 个循环节为(n+2)/2 的循环群，和 n/2 个循环节数为 n/2 的循环群。

综上，我们通过 Polya 定理就能计算出一共能做成多少种不同的项链。

4．代码（包含必要注释，采用最适宜阅读的 Courier New 字体，小五号，间距为固定值 12 磅）

```
#include<stdio.h>
#include<math.h>
int gcd(int a, int b)
{
  if(b == 0)
   return a;
  else return gcd(b, a % b);
}
int main()
{
   int n, m;
   while (scanf("%d%d", &n, &m), n | m)
   {
      int ans = 0;
      for (int i = 1; i <= m; i++)
         ans += pow(n, gcd(i, m));
      if (m & 1)
         ans += m * pow(n, m / 2 + 1);
      else
         ans += m / 2 * pow(n, m / 2) + m / 2 * pow(n, m / 2 + 1);
      ans /= m * 2;
      printf("%d\n", ans);
   }
   return 0;
}
```

5．思考与扩展：可以借鉴北大培训教材中做法。

1.1.1.1 题目 2

1.题目出处/来源

POJ 2154 Color

2.题目描述

给定 N 种颜色珠子，每种颜色的珠子的个数不限，将这些做成长度为 N 且不重复的项链(只考虑旋转，不考虑翻转)，并对所得结果 mod（p）处理。1<=N<=1000000000,1<=P<=30000

3.分析

这题和 2409 类似，不同之处在于，只考虑旋转，不考虑翻转；因此相对前面两个题目应该说是更简单，但一看数据范围，就不是这么回事了，2409 完全可以直接循环处理，但这题目 n 最大达 100000000，显然会 TLE，故需寻求更佳的解决方案。用欧拉函数进行优化：

旋转：顺时针旋转 i 格的置换中，循环的个数为 gcd（i，n），每个循环的长度为 n/gcd(i,n)。如果枚举旋转的格数 i，复杂度显然较高。有没有好方法呢？可以不枚举 i，反过来枚举 L。由于 L|N,枚举了 L，再计算有多少个 i 使得 0<=i<=n-1 并且 L＝gcd(i, n)。即 gcd（i,n）＝n/L。

不妨设 a = n/L=gcd(i, n), 不妨设 i = a*t 则当且仅当 gcd(L,t)=1 时 Gcd(i,n)=gcd(a*L,a*t)=a。因为 0<=i<n,所以 0<=t<n/a=L.所以满足这个条件的 t 的个数为 Euler(L).

4.代码

```
#include<iostream>
#include<cstring>
#include<cstdio>
using namespace std;
#define maxn 36000
int n, mod, ans;
int prim[35000];
bool flag[maxn + 20];

void get_prim() {
   memset(flag, 0, sizeof (flag));
   for (int i = 2; i <= 1000; i++)if (!flag[i])
          for (int j = i * i; j <= maxn; j += i)flag[j] = true;
   for (int i = 2, k = 0; i <= maxn; i++)
      if (!flag[i])prim[k++] = i;
}

int eular(int n) {
   int i = 0, ans = 1;
   for (i = 0; prim[i] * prim[i] <= n; i++) {
      if (n % prim[i] != 0)continue;
      ans *= prim[i] - 1;
      n /= prim[i];
      while (n % prim[i] == 0) {
         ans *= prim[i];
         n /= prim[i];
      }
   }
   if (n > 1)ans *= n - 1;
   return ans % mod;
}

int f(int c, int k, int mod) {
   int ans = 1;
   c = c % mod;
   while (k) {
      if (k & 1)ans = (c * ans) % mod;
      k >>= 1;
      c = (c * c) % mod;
   }
   return ans;
}

int main() {
   get_prim();
   int i, T;
   scanf("%d", &T);
   while (T-- && scanf("%d%d", &n, &mod)) {
      ans = 0;
      for (i = 1; i * i <= n; i++) {
         if (i * i == n)//枚举循环长度 l，找出相应的 i 的个数：gcd(i,n)=n/l.
            ans = (ans + f(n, i - 1, mod) * eular(i)) % mod;
         else if (n % i == 0)//有长度为 l 的循环，就会有长度为 n/l 的循环。
            ans = (ans + f(n, n / i - 1, mod) * eular(i) + eular(n / i) * f(n, i -
```

```
1, mod)) % mod;
            }
            printf("%d\n", ans);
        }
    }
```