SCAU

My code library

Acm/icpc 模板

XYM

2012/4/19

an AC a day keeps the WA away~

目录

常用技巧与知识

1.输入外挂

```
inline void scan(long long &k)//看情况改类型
{
        char c;
        while((c=getchar())&&(c>'9'||c<'0'));k=c-'0';
        while((c=getchar())&&(c<='9'&&c>='0'))k=k*10+c-'0';
}
```

用于有大量数字的输入；

2.Cin.push_back(ch);放回刚刚读到的字符；

3.读入 D1000 等数据时 ；

Scanf(“%s”,s); sscanf(“%c%d”,&ch,&num);

4,爆栈试试加上这句

```
#pragma comment(linker, "/STACK:102400000,102400000")
```

5.Printf 保留 n 位小数时，用 printf(“%.nf”,x)代替 printf(“%.nlf”,x)或用 long double 提高精度。(因为 float 类型参数会被提升为 double 类型，所以单单有%f 就是足够的,有些编译器不支持 lf)

7.要求输出 025 时，把输出格式变为%03d 即可，位宽为 3，不足用 0 填充，%3d 是位宽为 3，不足就填空格。

8.交换变量：swap(a,b); a^=b^=a^=b;（不建议使用）

9.返回 x 的整数部分，floor(x);减小误差 四舍五入 floor(x+0.5);四舍五入还可以直接%.0f.

10. 计 时 器 ： #include<ctime> printf(“Time used = %.2lf\n”,(double)clock() /CLOCKS_PER_SEC);输入输出的时间也算在里面了。

11.EOF　windows 下，Ctrl+z;　linux 下，Ctrl+d;

12.#define LOCAL　\\从文件中读入，和输出到文件

```
#ifdef  LOCAL
   Freopen(“ date.in ”,” r ”, stdin);
   Freopen(“ date.out ”,” w ”,stdout);
#endif
```

13.Sprintf(buf,”%d”,a);将信息输出到字符串。

14.大写字母和小写字母的转换

#include<ctype.h>　isalpha(c)判断 c 是否是字母；toupper(c)变成大写，tolower(c)变成小写。

15. 测试变量是否溢出 assert(x>=0),包含在#include<cassert>.

#define isok(x)　assert(x>=0) 宏定义方便。

16.下一个排列　while(next_permutation(p,p+n))应从最小排序开始得到所有排序.包含于 algorithm

17.printf 默认为右对齐，要左对齐只需要%后面加-即

Printf(“%-d”,a);

18.棋盘染色（黑白）gp[i][j]当(i+j)&1 为 1（奇数）的时候为白色，为 0（偶数）的时候为黑色。

数学

2.1 常用公式

(1). PI: Const double PI=2*acos(0);

(2). 求三角形面积：**A(x0,y0) B(x1,y1) C(x2,y2)**

Sabc=(x0*y1+x2*y0+x1*y2-x2*y1-x0*y2-x1*y0)/2;(A,B,C 成顺时针排列为正，逆时针为负，三点共线为 **0)**

(3). 斐波那契数列 **an=1/√5 [(1/2+√5/2)^ n-(1/2-√5/2)^n] (n=1,2,3.....)**（**√5** 表示根号 **5**）

(4). 指数的运算化为对数，求大数左边的数也可以用对数处理.

Lg(ab)=lg(a)+lg(b);

(5). 欧拉公式**:V** (顶点)**- E**(边数) **+ F**(面数) **= 2 ;**

(6). 斯特林数递推公式

(1).s[n][0]=0;s[n][n]=1;s[n][k]=s[n-1][k-1]+s[n-1][k]*(n-1);
解决问题**:**给你 **n** 个元素**,**求 **k** 个环排列的方法数**,dp** 思想注意理解.
(2). 定理 1:S[n][1]=S[n][n]=1;S[n][k]=S[n-1][k-1]+S[n-1][k]*k;
应用于求 **n** 个元素分成 **k** 个集合的方法数.
推广**:**

定理 2:对任何整数 $n \geq 1$ **和** $k \geq 0$ **，有**

$$S(n,k)=\frac{1}{k!}\sum_{i=0}^{k-1}(-1)^i C_k^i (k-i)^n$$ **（数学归纳法）**

定理 3 当 $1 \leq k \leq n \leq m$ **时，有**

$$m^n=\sum_{k=1}^{n} P_m^k S(n,k)=P_m^1 S(n,1)+P_m^2 S(n,2)+\cdots+P_m^n S(n,n)$$

P 表示全排列.

(7). 卡特兰数

1, 1, 2, 5, 14, 42, 132, 429, 1430, 4862, 16796, 58786

(1):h(n)= h(0)*h(n-1)+h(1)*h(n-2) + ... + h(n-1)h(0) (其中 n>=2)
(2): h(n)=h(n-1)*(4*n-2)/(n+1);
(3): h(n)=C(2n,n)/(n+1) (n=1,2,3,...)

应用：括号化问题，出栈次序问题，凸多边形的三角剖分问题，用给定节点组成二叉树的问题.

(8).pick 定理（以后补充其他）

定理一：**Pick** 定理，**1899** 年

设 **Γ** 为平面上以格子点为顶点之单纯多边形，则其面积为

$$A = \frac{b}{2} + i - 1 \qquad (8)$$

其中 ***b*** 为边界上的格子点数，***i*** 为内部的格子点数。**(8)**式叫做 **Pick** 公式。

(9) 平方和：**1^2+2^2+3^2+……+n^2=n*(n+1)*(2*n+1)/6;**

立方和：**1^3+2^3+3^3+……+n^3=(n*(n+1)/2)^2=n^2*(n+1)^2/4;**

(10). 向量 **AxB>0** 则 **A** 在 **B** 的顺时针方向，**AxB<0** 则 **A** 在 **B** 的逆时针方向，**AxB=0,**向量 **A** 和 **B** 共线，方向相同或相反。

(11).保留小数的时候要注意四舍五入时**-0.004~-0.00**，**-0.005~-0.01**；这里的**-0.00** 要去掉负号，否则会 **wa**。

(12).本原勾股数:**a,b,c(gcd(a,b,c)=1),** 那么假设 **c+b=s^2,c-b=t^2**，则有 **c=(s^2+t^2)/2,b=(s^2-t^2)/2**，**a=st**；（**t** 由给定的边长可由别的未知数表示）。

(13).某一天是周几蔡乐公式：**W=[C/4]-2C+y+[y/4]+[26(m+1)/10]+d-1** （其中**[ ]** 为取整符号）

其中,**W** 是所求日期的星期数。

如果求得的数大于 **7**，可以减去 **7** 的倍数，直到余数小于 **7** 为止。

如果求得的数小于 **7**，可以加上 **7** 的倍数，知道大于 **0** 小于 **7** 为止。

c 是公元年份的前两位数字，**y** 是已知公元年份的后两位数字；**m** 是月数，**d** 是日数。

方括**[ ]**表示只截取该数的整数部分。

还有一个特别要注意的地方：所求的月份如果是 1 月或 2 月，则应视为前一年的 13 月或 14 月，所以公式中 m 的取值范围不是 1-12，而是 3-14。

最后还有一点，公元 1582 年罗马教皇曾经下令修改历法，这一年的 10 月 4 日(星期 4)的下一天改为 10 月 15 日。所以在 1582 年以前的日子，不能加以套用。

(14).位运算

K=k&(k-1)去除末尾的 1，可用于计算 k 中有几个 1；
K=k&(-k)返回该数的最后的一个 1 所表示的最小幂，树状数组的使用。
K^b^c^b=k^b 异或的交换律。

(15) n!=(n/e)*n * sqrt(2*pi*n)

```
//n 的阶乘中素因子 a 的幂
int aofn( int a,int n )
{
    int count = 0;
    while( n )
    {
        n/=a; count += n;
    }
    return count;
}
```

(16) 十进制表示整数的整除判别法。

（1）.如果一个数的末位数能别 2 或 5 整除，那么这个数能被 2 或 5 整除。
（2）.如果一个整数的末两位数能被 4 或 25 整除，那么这个数就能被 4 或 25 整除。
（3）.如果一个整数的末三位数能被 8 或 125 整除，那么这个数就能被 8 或 125 整除.
（4）.如果一个数各数位上的数字和能被 9 或 3 整除，那么这个数就能被 3 或 9 整除。
（5）.如果一个整数的末三位数与末三位以前的数字所组成的数的差能被 7 或 11 或 13 整除，那么这个数就能被 7 或 11 或 13 整除。
（6）.如果一个整数的奇数位上的数字和与偶数位上的数字和之差能被 11 整除，那么这个数能被 11 整除。
（7）.0 能被任何数整除，+，-1 能整除任何数。

(17).费马大定理 n>2,x^n+y^n=z^n,这个方程无解。

(18).威尔逊定理 若 p 为素数，(p-1)！=-1（mod p），若结论成立也能证明 p 是素数。

(19).费马小定理：假如 p 是质数，且(a,p)=1，那么 a^(p-1) ≡1（mod p） 假如 p 是质数，且 a,p 互质，那么 a 的(p-1)次方除以 p 的余数恒等于 1；
运用：取模以及测试素性

b^(p-1）≡1(mod p)

则 p 必定是一个质数。

这个定理告诉我们，判定一个数是否为质数，没有必要测试所有的小于 p 的自然数，只要测试所有的小于 p 的质数就可以了。事实上，测试小于 p 的平方根的所有质数就可以了.**根据这个性质可以求逆元 b 的逆元是 b^(p-2)(p 是模数)。**

（20）.二次剩余：数论基本概念之一。若 a、m 的最大公约数为 1(a 可以大于 m)〔记为(a，m）＝1)，m 整除（x^2－a）〔记为 x^2≡ a（mod m)〕有解，则称 a 为模 m 的二次剩余（或平方剩余)； 否则，称 a 为模 m 二次非剩余（或平方非剩

$$\left(\frac{a}{p}\right)=\begin{cases}1 & a \text{ 是模 } P \text{ 二次剩余；}\\ -1 & a \text{ 是模 } P \text{ 二次剩余；}\\ 0 & P\mid a\end{cases}$$

二次剩余

余)。解一般二次同余式 ax2＋bx＋c≡0（mod m）的问题可归结为解 x^2≡n（mod m）问题（见同余)。欧拉给出了判别条件：若 p 是奇素数，(a，p）＝1，则 a 是模 p 的二次剩余的充分必要条件为 a ^ ((p - 1) / 2)≡1（mod p ）；a 是模 p 的二次非剩余的充分必要条件为 a ^ ((p - 1) / 2)≡-1（modp)。称｛k｜0＜k≤m，(k，m）＝1｝为 m 的 简化剩余系。

2.2 欧几里得算法

```
LL gcd(LL a,LL b)
{
        return b==0?a:gcd(b,a%b);
}
LL kgcd( LL a,LL b )
{
        if(!a)return b; if(!b)return a;
        if( !(a&1) && !(b&1) )
                return kgcd(a/2,b/2)<<1;
        else if( !(b&1) )
                return kgcd( a,b>>1 );
        else if( !(a&1) )
                return kgcd( a>>1,b );
        else return kgcd( abs(a-b),min(a,b) );
}
//计算机中的负数模(a%n)与数论中的不一样，需要转换
```

```
LL rightmod(LL a,LL n)
{
        return (n+a%n)%n;
}
```

2.3 拓展欧几里得

把这个实现和 **Gcd** 的递归实现相比，发现多了下面的 **x,y** 赋值过程，这就是扩展欧几里德算法的精髓。

可以这样思考**:**

对于 **a' = b, b' = a % b** 而言，我们求得 **x, y** 使得 **a'x + b'y = Gcd(a', b')**

由于 **b' = a % b = a - a / b * b (**注：这里的**/**是程序设计语言中的除法**)**

那么可以得到**:**

a'x + b'y = Gcd(a', b') ===>

bx + (a - a / b * b)y = Gcd(a', b') = Gcd(a, b) ===>

ay +b(x - a / b*y) = Gcd(a, b)

因此对于 **a** 和 **b** 而言，他们的相对应的 **p**，**q** 分别是 **y** 和**(x-a/b*y)**；

p，**q** 分别为 **ax+by=gcd(a,b)**解出的 **x**，**y**；

补充：关于使用扩展欧几里德算法解决不定方程的办法

对于不定整数方程 **xa+yb=c**，若 **c mod Gcd(a, b)=0,**则该方程存在整数解，否则不存在整数解。

上面已经列出找一个整数解的方法，在找到 **x * a+y * b = Gcd(a, b)**的一组解 **x0,y0** 后，**x * a+y * b = Gcd(a, b)**的其他整数解满足：

x= x0 + b/Gcd(a, b) * t

y = y0 - a/Gcd(a, b) * t(其中 **t** 为任意整数**)**

(原方程**(ax+by=c)**的解可以表示为**)**

X（原）**=x0* c/Gcd(a, b)) + b/Gcd(a, b) * t**

y（原）**=y0* c/Gcd(a, b)) - a/Gcd(a, b) * t**

最小整数解**(((x0* c/Gcd(a, b))%b1 + b1)%b1 (b1= b/Gcd(a, b));**

((y0* c/Gcd(a, b))%a1 + a1)%a1 (a1= a/Gcd(a, b));

终极模板(可解中国剩余定理互质和不互质的情况)

```
#include<cstdio>
#include<string>
#include<cstring>
#include<cmath>
#include<iostream>
#include<algorithm>
#include<map>
```

```
#include<queue>
#include<vector>
#define LL long long
using namespace std;
const int size=100005,MAX=1<<30,MIN=-(1<<30);
LL m[10],r[10];
//拓展欧几里德
void ex_gcd(LL a,LL b,LL &d,LL &x,LL &y)//解 a*x+b*y=gcd(a,b)=d
{
        if(b==0)
        {
                x=1,y=0,d=a;
        }
        else
        {
                ex_gcd(b,a%b,d,y,x);
                y-=x*(a/b);
        }
}
LL gcd(LL a,LL b)
{
        return b==0?a:gcd(b,a%b);
}
LL china_remain(int n)//解模线性方程组
{
        LL a,b,c,c1,c2,x,y,d,N;
        a=m[0];c1=r[0];
         if(c1==0)c1=m[0];
        for(int i=1;i<n;i++)
        {
                b=m[i];c2=r[i];
                ex_gcd(a,b,d,x,y);
                c=c2-c1;
                if(c%d)return -1;
                LL b1=b/d;
                x=((c/d*x)%b1+b1)%b1;//最小整数解
                c1=a*x+c1;//新方程的 c1
                a=a*b1; //新方程的 a
        }
        return c1;
}
int main()
{
        int T,n;
```

```
        scanf("%d",&T);
        for(int t=1;t<=T;t++)
        {
                scanf("%d",&n);
                for(int i=0;i<n;i++)
                        scanf("%lld",&m[i]);
                for(int i=0;i<n;i++)
                        scanf("%lld",&r[i]);
                cout<<"Case "<<t<<": "<<china_remain(n)<<endl;
        }
        return 0;
}
```

2.4 二分快速幂

```
//求 b=a^k%M
LL fastmod(LL a,LL k,LL M)
{
        LL b = 1;
        while( k )
        {
                if( k&1 ) b = a*b%M;
                a = (a%M)*(a%M)%M;
                k /= 2;
        }
        return b;
}
```

2.5 欧拉函数

```
long long phi[3000005]={0};
void Euler(int n)//n 为范围
{
        phi[1]=1;
        for(int i=2;i<=n;i++)
        {
                if(!phi[i])
                {
                        for(int j=i;j<=n;j+=i)
                        {
```

```
                                if(!phi[j])phi[j]=j;
                                phi[j]=phi[j]/i*(i-1);
                            }
                        }
                }
}
```

利用 E(k) =k*(1-1/p1)*(1-1/p2)…(1-1/pk)(pi 表示第 i 个素因子)
从素因子开始更新后面的数。

2.

在程序中利用欧拉函数如下性质，可以快速求出欧拉函数的值(a 为 N 的质因素)
(1) 若(N%a==0 && (N/a)%a==0) 则有:E(N)=E(N/a)*a;
(2) 若(N%a==0 && (N/a)%a!=0) 则有:E(N)=E(N/a)*(a-1);
这种方法也可，但是编程复杂度较高，一般不用。
欧拉函数：
定义：　　对于正整数 n，ϕ(n)是小于或等于 n 的正整数中，与 n 互质的数的数目；
　　例如：ϕ(8) = 4，因为 1，3，5，7 均和 8 互质。
性质：　1.　若 p 是质数，ϕ（p）= p-1.
　　2.　若 n 是质数 p 的 k 次幂，ϕ（n）= （p-1）p^(k-1)
　　因为除了 p 的倍数都与 n 互质
　　3.　欧拉函数是积性函数，若 m，n 互质，ϕ（mn）= ϕ(m)ϕ(n)
　　根据这 3 条性质我们就可以退出一个整数的欧拉函数的公式，因为一个数总可以一些质数的乘积的形式。
　　E(k) = (p1-1)(p2-1)…(pi-1)*(p1^(a1-1))(p2^(a2-1))…(pi^(a i-1))
　　= k*(p1-1)(p2-1)…(pi-1)/(p1*p2*…pi)
　　= k*(1-1/p1)*(1-1/p2)…(1-1/pk)
在程序中利用欧拉函数如下性质，可以快速求出欧拉函数的值(a 为 N 的质因素)
若(N%a==0 && (N/a)%a==0) 则有:E(N)=E(N/a)*a;
若(N%a==0 && (N/a)%a!=0) 则有:E(N)=E(N/a)*(a-1);

当 n 很大时不能预处理，只能直接求 E（x）;
由下面两个性质
1. 若 n 是质数 p 的 k 次幂，ϕ（n）= （p-1）p^(k-1)
2. 若 m，n 互质，ϕ（mn）= ϕ(m)ϕ(n)

```
int p[10005],pos=0;bool vd[10005];
void prime(int n)//筛素数到最大的 sqrt
{
    vd[0]=vd[1]=1;
    for(int i=2;i<=n;i++)
            if(!vd[i])
            {
                    p[pos++]=i;
                    for(int j=i*2;j<=n;j+=i)
```

```
                                vd[j]=1;
                }
}
long long Euler(long long n)
{
    long long ans=1;
    for(int i=0;i<pos&&p[i]<=n;i++)
    {
            if(n%p[i]==0)//由幂公式
            {
                    n/=p[i];ans*=p[i]-1;
                    while(n%p[i]==0)
                            ans*=p[i],n/=p[i];
            }
    }
    if(n>1)ans*=n-1;//n 不为 1 只能为大于 10000 的素数由积公式乘上即可
    return ans;
}
```

拓展问题：

（1） 求小于 n 且与 n 互质的所有数的总和：gcd(n,i)=1,gcd(n,n-i)=1，sum=phi(n)/2*n;

2.6 质因子分解及约数

```
//count 表示质因子个数,d[i][0]和 d[i][1]分别表示第 i 个质因子和个数
LL d[100][2];
LL divprime(LL n)
{
    LL i,count=0,k=n;
    for( i = 2; i*i <= k; i++ )
        if( n%i == 0 )
        {
            d[count][0] = i; d[count][1] = 0;
            while( n%i == 0 )
                n/=i,d[count][1]++; count++;
        }
        if( n > 1 )
            d[count][0] = n,d[count][1] = 1,count++;
        return count;
}
//搜索质因子的各种组合，即找到 n 所有约数
//调用方式 finddiv(0,1,n,count)
//count 为因子个数,变量 ji 保存的是每次搜到的约数
```

```
void finddiv( LL ceng,LL ji,LL n,LL count )
{
    if( ceng == count )
    {
        cout<<ji<<endl; return;
    }
    LL i,m;
    for( i = 0,m = 1; i <= d[ceng][1] ; i++ )
        finddiv( ceng+1,ji*m,n,count),m *= d[ceng][0];
}
```

2.7 素数线性筛法+质因子个数和

```
//count 表示质因子个数,d[i][0]和 d[i][1]分别表示第 i 个质因子和个数
const int Size = 1000000; bool flag[Size+1];//是否为素数
int prime[1000001]; int sum[Size+1];//n 的所有质因子的和
void xianxingshai(void)
{
    int count,i,j;
   for(i=0;i<=Size;i++)
   flag[i]=true,sum[i]=0;
    flag[0] = flag[1] = false;
    for( count = 0, i = 2; i <= Size; i++ )
    {
        if( flag[i] )
        {
            prime[count++] = i;
            sum[i] = i;
        }
        for( j = 0; j < count && i*prime[j] <= Size; j++ )
        {
            flag[ i*prime[j] ] = false;
            if( i%prime[j] == 0 )
            {
                sum[ i*prime[j] ] = sum[i];break;
            }
            else sum[ i*prime[j] ] = sum[ i ] + prime[j];
        }
    }
}
```

2.8 快速筛素数

```
const int maxsz=2147483647;//检测的数值上限
```

```
const int maxsz2=maxsz/2;
const int maxsqrt=sqrt(double(maxsz))/2;
char a[maxsz/16+2]; //为了提速,当保证 n 为奇数时,(n&1)||(2==n)可以省略;当保证 n!=2 时,(2==n)可以省略
int prime[maxsz/16+2],cnt;

bool isprime(int n)
{
    return ((n&1)||(2==n))&&(a[(n)>>4]&(1<<(((n)>>1)&7)));
}

void sieve()
{
    cnt=0;
    memset(a,255,sizeof a);
    a[0]=0xFE;
    for(int i=1;i<maxsqrt;i++)
    {
        if(a[i>>3]&(1<<(i&7)))
            for(int j=i+i+i+1;j<maxsz2;j+=i+i+1)
                a[j>>3]&=~(1<<(j&7));
    }
    for(int i=1;i<maxsz;i++)
        if(isprime(i))
            prime[cnt++]=i;
    printf("%d %d\n",cnt,prime[cnt-1]);
}
```

2.9 miller rabin 素数测试

```
LL witness( LL a,LL n )
{
    LL u = n-1,t = 0;
    while( !(u&1) )
    {
        u>>=1;t++;
    }
    int x1,x0 = fastmod(a,u,n); //二分快速幂
    for( int i = 1; i <= t; i++ )
    {
        x1 = x0*x0%n;
        if( x1 == 1 && x0 != 1 && x0 != n-1 )
            return 1;
```

```
        x0 = x1;
    }
    if( x0 != 1 ) return 1;
    else return 0;
}
LL miller_rabin(LL n ,LL s )
{
    //s 为测试次数
    if( n==2 ) return 1;
    if( n%2==0 || n < 2 ) return 0;
    int j,a;
    for( j = 1; j <= s; j++ )
    {
        a = rand()*(n-1)/RAND_MAX + 1;
        if( witness(a,n) ) return 0;
    }
    return 1;
}
```

2.10 康拓展开

（1）康拓展开

所谓康拓展开是指把一个整数 X 展开成如下形式：

X=a[n]*(n-1)!+a[n-1]*(n-2)!+...+a[i]*(i-1)!+...+a[2]*1!+a[1]*0!。（其中，a 为整数，并且 0<=a[i]<i(1<=i<=n)）

（2）应用实例

{1,2,3,4,...,n}表示 1,2,3,...,n 的排列如 **{1,2,3}** 按从小到大排列一共 **6** 个：**123 132 213 231 312 321**。他们间的对应关系可由康托展开来找到。

1324 是**{1,2,3,4}**排列数中第几个大的数：

第一位是 **1** 小于 **1** 的数没有，是 **0** 个 **0*3!** ；

第二位是 **3** 小于 **3** 的数有 **1** 和 **2**，但 **1** 已经在第一位了，即 **1** 未出现在前面的低位当中，所以只有一个数 **2 1*2!** ；

第三位是 **2** 小于 **2** 的数是 **1**，但 **1** 在第一位，即 **1** 未出现在前面的低位当中，所以有 **0** 个数 **0*1!** ；

所以比 **1324** 小的排列有 **0*3!+1*2!+0*1!=2** 个，**1324** 是第三个大数。

```
long int fac[]={1,1,2,6,24,120,720,5040,40320,362880}; //表示阶乘的结果
int a[100]={4,5,3,2,1};
 int Cantor(int len)//返回有多少序列小于 a 序列
 {
     int i,j,num,temp;
     num=0;
```

```
    for(i=0;i<len;i++)
    {
        temp=0;     //temp 记录当前数位前面的低数位中小于当前位数上的数字的个数
        for(j=i+1;j<len;j++)
            if(a[j]<a[i])
                temp++;
        num+=fac[len-1-i]*temp;  //乘以相应的阶乘
    }
    return num;
}
```

如何判断给定一个位置，输出该位置上的数列，康拓展开的逆运算，例如：

{1,2,3,4,5}的全排列，并且已经从小到大排序完毕，请找出第 **96** 个数：

首先用 **96-1** 得到 **95**

用 **95** 去除 **4!** 得到 **3** 余 **23**，即有 **3** 个数比该数位上的数字小，则该数位的数字为 **4**；

用 **23** 去除 **3!** 得到 **3** 余 **5**，即有 **3** 个数比该数位上的数字小，理应为 **4**，但 **4** 已在前面的高位中出现过，所以该数位的数字为 **5**；

用 **5** 去除 **2!**得到 **2** 余 **1**，即有 **2** 个数比该数位上的数字小，则该数位的数字为 **3**；

用 **1** 去除 **1!**得到 **1** 余 **0**，即有 **1** 个数比该数位上的数字小，则该数位的数字为 **2**；

最后一个数只能是 **1**；

所以这个数是 **45321**

```
int res[100];
void CantorReverse(int num,int len)//排第 num 的长度为 len 的序列保存在 res 里
{
    num--;     //勿丢
    int i,j;
    bool hash[10]={0};
    for(i=0;i<len;i++)
    {
        int tmp=num/fac[len-1-i];  //tmp 表示有 tmp 个数字比当前位置上的数字小
        for(j=0;j<=tmp;j++)
            if(hash[j]) tmp++;
        res[i]=tmp+1;
        hash[tmp]=1;
        num%=fac[len-1-i];
    }
    return;
}
```

2.11 高斯消元

高斯消元

题意：问 n+1 个 n 维点的球心，即其他点到该点距离相等

设距离为 R,很容里列出 n+1 个方程，但是有 2 次项，只要下一行减上一行即可

得到一个 n 个方程 n 个未知数，用高斯消元

矩阵为

A[i][j] = (V[i+1][j] - V[i][j])*2

A[i][n+1] = V[i+1][j]^2 - V[i][j]^2 （只是增广的那一项）

会超 long long 但用 java 高精度会超时！

1 <= N <= 50, |xi| <= 10^17 而且答案是整数解且 |Xi| <= 10^17

只要运算过程中取模一个大于 2*10^17 的素数即可

Ax = B

=> Ax %p = B %p

=> A%p x = B %p

即原方程的解等价于系数为 A%p 的方程的解

注意：

由于答案可以是负数，%p 的话结果是[0,p)，不对

所以一开始对所有坐标都平移一下，都加上 10^17，最后答案减去 10^17 即正确了

我自己写的高斯消元超时，太慢了

他这种是对 row 这一行都除以 A[row][row]使得 A[row][row]为 1，很快

由于取余，所以可以直接除（乘以逆元），不怕整不整除问题!!

--yuan

```
#define Lg long long
using namespace std;
const Lg mod=10000000000000003LL,inc=10000000000000000LL;
Lg A[51][51];
int n,m;
inline Lg MOD(Lg a)
{
        if(a<0) return a%mod+mod;
        else if(a>=mod) return a%mod;
        else return a;
}
inline Lg multiply(Lg a,Lg b)//大数 a*b，防止溢出，转变成快速二分求和，非递归算法
{
        Lg ans=0,tem=MOD(a);
        while(b)
        {
```

```
                if(b&1)
                {
                        ans=ans+tem;
                            if(ans>=mod)ans-=mod;
                }
                tem=tem<<1;if(tem>=mod)tem-=mod;
                b>>=1;
        }
        return ans;
}
inline Lg exp(Lg a,Lg b)//a^b 非递归写法
{
        Lg ans=1;
        while(b)
        {
                if(b&1)ans=multiply(ans,a);
                a=multiply(a,a);b>>=1;
        }
        return ans;
}
void Gauss()//高斯消元
{
        for(int k=0;k<n;k++)
        {
                Lg max_a=A[k][k];int bin=k;
                for(int i=k+1;i<n;i++)//找到当前系数最大的放到最上位置
                        if(A[i][k]>max_a)
                                max_a=A[i][k],bin=i;
                if(bin!=k)
                {
                        for(int i=k;i<=n;i++)
                                swap(A[bin][i],A[k][i]);
                }
                Lg Ni=exp(A[k][k],mod-2LL);//取逆元，方便乘，同时也使当前最
大行系数为 1，方便消元。
                for(int j=k+1;j<=n;j++)
                {
                        A[k][j]=multiply(A[k][j],Ni);
                        for(int i=k+1;i<n;i++)
                                A[i][j]=MOD(A[i][j]-multiply(A[k][j],A
[i][k]));
                }
        }
        for(int i=n-1;i>=0;i--)//计算 xi，存在 A[i][i]中
```

```
        {
                for(int j=n-1;j>i;j--)
                        A[i][n]=MOD(A[i][n]-multiply(A[i][j],A[j][j]));
                A[i][i]=A[i][n];
        }
}
int main()
{
        int T;scanf("%d",&T);
        for(int t=1;t<=T;t++)
        {
                scanf("%d",&n);
                for(int i=0;i<=n;i++)
                        for(int j=0;j<n;j++)
                        {
                                scanf("%I64d",&A[i][j]);
                                A[i][j]+=inc;
                        }
                        for(int i=0;i<n;i++)
                        {
                                A[i][n]=0;
                                for(int j=0;j<n;j++)
                                {
                                        A[i][n]=MOD(A[i][n]+multiply(A[i
+1][j],A[i+1][j])-multiply(A[i][j],A[i][j]));
                                        A[i][j]=MOD(2LL*(A[i+1][j]-A[i]
[j]));
                                }
                        }
                        Gauss();
                        printf("Case %d:\n",t);
                        for(int i=0;i<n;i++)
                        {
                                printf("%I64d",A[i][i]-inc);
                                if(i!=n-1)printf(" ");else puts("");
                        }
        }
        return 0;
}
```

高斯消元解决开灯问题

题意：给出 n 个灯，m 个开关，某个开关能同时控制某些灯 。 问达到最终状态的方案数

消元，答案即为 2^x x 为自由变量的个数或无解

设开关变量分别为 x1,x2…,xm 控制矩阵为 A A[i,j]=1 表示灯 i 被开关 j 控制

Y0+AX' = Y n 个方程，m 个未知量 X'为 X 的列向量形式 Y0 是初始状态

POJ 1830 就有给出了初始状态，只要 Y^Y0 作为目标状态即可

标程是只消去一次，然后每次读入询问再检查

我不会，就每次读入询问都消元一次…

——yuan

方程为 A*X1^B*X2^C*X3^D*X4=sta;（A，B。。。。表示是否和这个开关有关，0 表示无关，1 表示有关）

.....

```
#define LG long long
using namespace std;
const int maxn=10005,MAX=1<<30,MIN=-(1<<30);
bool A[55][55],B[55][55];
int n,m;
LG Guass()//高斯消元
{
        int rew=0,flag;
        for(int j=0;j<m;j++)
        {
                flag=-1;
                for(int i=rew;i<n;i++)//寻找当前列系数不为 0 项
                        if(A[i][j]&&(flag=i,1))break;
                        if(flag==-1)continue;
                        for(int i=j;i<=m;i++)//找到就交换该列
                                swap(A[flag][i],A[rew][i]);
                        for(int i=rew+1;i<n;i++)//对后面的方程消元
                                if(A[i][j])
                                        for(int k=j+1;k<=m;k++)
                                                A[i][k]=A[i][k]^A[rew]
[k];
                        rew++;//下一行
        }
        for(int i=rew;i<n;i++)//查找是否存在 0 0 0 0 0 1 这种不合法的解，即无
解。
                if(A[i][m])return 0;
        return 1LL<<(m-rew);//m-rew 为变元数
}
int main()
{
```

```
        int T;
        scanf("%d",&T);
        for(int t=1;t<=T;t++)
        {
                scanf("%d%d",&n,&m);
                memset(B,0,sizeof(B));
                for(int i=0;i<m;i++)
                {
                        int num,x;scanf("%d",&num);
                        for(int j=0;j<num;j++)
                        {
                                scanf("%d",&x);
                                B[x-1][i]=1;
                        }
                }
                printf("Case %d:\n",t);
                int q;scanf("%d",&q);
                for(int k=0;k<q;k++)
                {
                        memcpy(A,B,sizeof(B));
                        for(int i=0;i<n;i++)
                                scanf("%d",&A[i][m]);
                        printf("%I64d\n",Guass());
                }
        }
        return 0;
}
```

图论

3.1 最短路(dij,spfa,floyd)

Dij

```
struct node
{
        int v,w;
        bool operator < (const node a)const
        {
                return a.w<w;
```

```
        }
};
void dij(int st)
{
        priority_queue< node >q; node in,out;
        for(int i=0;i<=n;i++)
                d[i]=INF,vd[i]=0;
        in.w=d[st]=0;in.v=st;q.push(in);
        while(!q.empty())
        {
                out=q.top();q.pop();
                if(vd[out.v])continue;
                vd[out.v]=1;
                for(int i=h[out.v];i>=0;i=eg[i].next)
                {
                        int u=eg[i].v,w=eg[i].w;
                        if(!vd[u]&&d[u]>d[out.v]+w)
                                in.w=d[u]=w+d[out.v],in.v=u,q.push(in);
                }
        }
}
```

SPFA 复杂度(O(Km))

```
int spfa(int st,int fn)
{
        queue< int > q;
        for(int i=0;i<=n;i++)
                d[i]=INF,vd[i]=0,cnt[i]=0;
        d[st]=0;vd[st]=1;q.push(st);
        while(!q.empty())
        {
                int u=q.front();
                vd[u]=0;q.pop();
                for(int i=head[u];i>=0;i=eg[i].next)
                {
                        int v=eg[i].v,w=eg[i].w;
                        if(d[v]>d[u]+w)
                        {
                                d[v]=d[u]+w;
                                if(!vd[v])
                                {
                                        vd[v]=1;
                                        cnt[v]++;
```

```
                                        q.push(v);
                                        if(cnt[v]>n)return -1;
                                    }
                                }
                            }
                        }
        if(d[fn]==INF)return -1;
        return d[fn];
}
```

优点：可以处理负回路，当一个点入队超过 n 次有负回路(cnt[v]>n)

FLOYD

```
const int Size=1000,INF=1<<30;
int gp[Size][Size];//构图时初始化为 INF，gp[i][i]=0;
int Floyd(int from,int to,int n)
{
    for(int i=0; i<=n; i++)
        for(int k=0; k<=n; k++)
            for(int j=0; j<=n; j++)
                if(gp[k][i]!=INF&& gp[i][j]!=INF&& gp[k][i] + gp[i][j] <
gp[k][j] )
                    gp[k][j] = gp[k][i] + gp[i][j];
    return gp[from][to];
}
```

次短路（朴素算法）

```
void dij(int st,int fn)（适用于有向无环图）
{
        for(int i=0;i<=n;i++)
        {
                d[i][0].v=d[i][1].v=i;d[i][0].kind=0;d[i][1].kind=1;
                d[i][0].w=d[i][1].w=MAX;
                vd[i][0]=vd[i][1]=cnt[i][0]=cnt[i][1]=0;
        }
        d[st][0].w=0;cnt[st][0]=1;
        while(1)
        {
                int flag,Min=MAX,bin;
               for(int i=0;i<n;i++)//找到当前最短的深度
                            if(!vd[i][0]&&d[i][0].w<Min)
                                    flag=0,Min=d[i][0].w,bin=i;
                            else if(!vd[i][1]&&d[i][1].w<Min)
```

```
                                flag=1,Min=d[i][1].w,bin=i;
                    if(Min==MAX)break;//全部更新完成，退出
                    vd[bin][flag]=1;
                    for(int i=h[bin];i>=0;i=eg[i].next)
                    {
                            int u=eg[i].v,w=eg[i].w+d[bin][flag].w;
                            if(!vd[u][0]&&d[u][0].w>w)
                            {
                                    if(d[u][0].w!=MAX)
                                            d[u][1].w=d[u][0].w,cnt[u][1]
=cnt[u][0];
                                    d[u][0].w=w,cnt[u][0]=cnt[bin][flag];
                            }
                            else if(!vd[u][0]&&d[u][0].w==w)
                                    cnt[u][0]+=cnt[bin][flag];
                            else if(!vd[u][1]&&d[u][1].w>w)
                                    d[u][1].w=w,cnt[u][1]=cnt[bin][flag];
                            else if(!vd[u][1]&&d[u][1].w==w)
                                    cnt[u][1]+=cnt[bin][flag];
                    }
            }
            printf("%d %d\n",d[fn][1].w,cnt[fn][1]);
}
```

若为无向图，可以先求最短路，然后枚举删除最短路上的边，求次短路。

3.2 最小生成树

Kruskal

```
struct edge
{
        int x,y,w;
        bool operator < (const edge a)const
        {
                return a.w<w;
        }
};
int fa[1005];
int find(int n)
{
        return n==fa[n]?n:fa[n]=find(fa[n]);
}
void init(int n)//并查集查找
```

```
{
        for(int i=0;i<=n;i++)
                fa[i]=i;
}
int Kru(int n)
{
        priority_queue <edge> q;
        edge e;
        int len=(n-1)*n/2;
        for(int i=0;i<len;i++)//读入边进优先队列
        {
                scanf("%d%d%d",&e.x,&e.y,&e.w);
                q.push(e);
        }
        int cnt=0,ans=0;
        while(!q.empty()&&cnt<n-1)
        {
                e=q.top();q.pop();
                int a=find(e.x),b=find(e.y);
                if(a!=b)
                {
                        fa[a]=b;
                        cnt++;
                        ans+=e.w;
                }
        }
        return ans;
}
```

3.3 二分图

匈牙利 Maxmatch (临接表 O(n*m))

```
int dfs(int st)
{
        for(int i=head[st];i>=0;i=eg[i].next)
        {
                int u=eg[i].v;
                if(!vd[u])
                {
                        vd[u]=1;
                        if(mct[u]==-1||dfs(mct[u]))
                        {
```

```
                                mct[u]=st;return 1;
                            }
                        }
                }
        return 0;
}
void Maxmatch(int n)//匈牙利算法
{
        int ans=0;
        memset(mct,-1,sizeof(mct));
        for(int i=1;i<=n;i++)
        {
                memset(vd,0,sizeof(vd));
                ans+=dfs(i);
        }
        printf("%d\n",n-ans);
}
```

最小点覆盖（与点相连的边被覆盖）=最大匹配；
最大独立集（集合内的点没有边相连）=n-最大匹配数；
最小路径覆盖（有向连通无回路图最少边覆盖全部点，点不重复覆盖，每个点只能是一条路径的起点和另一条的终点）=n-最大匹配数；
对二分图的一个点集求匹配数就是最大匹配数，若不能先分成二部图则，对无向图求匹配，所得结果除 2 即为匹配数。

独立集：

独立集是指图的顶点集的一个子集,该子集的导出子图不含边.如果一个独立集不是任何一个独立集的子集，那么称这个独立集是一个极大独立集.一个图中包含顶点数目最多的独立集称为最大独立集。最大独立集 一定是极大独立集，但是极大独立集不一定是最大的独立集。

支配集：

与独立集相对应的就是支配集，支配集也是图顶点集的一个子集，设 **S** 是图 **G** 的一个支配集，则对于图中的任意一个顶点 **u**，要么属于集合 **s**，要么与 **s** 中的顶点相邻。 在 **s** 中除去任何元素后 **s** 不再是支配集，则支配集 **s** 是极小支配集。称 **G** 的所有支配集中顶点个数最 少的支配集为最小支配集，最小支配集中的顶点个数成为支配数。

最小点的覆盖：

最小点的覆盖也是图的顶点集的一个子集，如果我们选中一个点，则称这个点将以他为端点的所有边都覆盖了。将图中所有的边都覆盖所用顶点数最少，这个集合就是最小的点的覆盖。

最大团：

图 G 的顶点的子集，设 D 是最大团，则 D 中任意两点相邻。若 u，v 是最大团，则 u,v 有边相连，其补图 u,v 没有边相连，所以图 G 的最大团=其补图的最大独立集。

一些性质：

最大独立集+最小覆盖集=V

最大团=补图的最大独立集

最小覆盖集=最大匹配

KM

```
const int MAX = 1024;
int n; // X 的大小
int weight [MAX] [MAX]; // X 到 Y 的映射（权重）
int lx [MAX], ly [MAX]; // 标号
bool sx [MAX], sy [MAX]; // 是否被搜索过
int match [MAX]; // Y(i) 与 X(match [i]) 匹配
// 初始化权重
void init (int size);
// 从 X(u) 寻找增广道路，找到则返回 true
bool path (int u);
// 参数 maxsum 为 true ，返回最大权匹配，否则最小权匹配
int bestmatch (bool maxsum = true);
void init (int size)
{
    // 根据实际情况，添加代码以初始化
    n = size; for (int i = 0; i < n; i ++)
        for (int j = 0; j < n; j ++)
            scanf ("%d", &weight [i] [j]);
}
bool path (int u)
{
    sx [u] = true;
    for (int v = 0; v < n; v ++)
        if (!sy [v] && lx[u] + ly [v] == weight [u] [v])
        {
            sy [v] = true;
            if (match [v] == -1 || path (match [v]))
            {
                match [v] = u;
            return true;
            }
```

```
        }
        return false;
}
int bestmatch (bool maxsum)
{
    int i, j;
    if (!maxsum)
    {
        for (i = 0; i < n; i ++)
            for (j = 0; j < n; j ++)
                weight [i] [j] = -weight [i] [j];
    }
    // 初始化标号
    for (i = 0; i < n; i ++)
    {
        lx [i] = -0x1FFFFFFF;
        ly [i] = 0;
        for (j = 0; j < n; j ++)
            if (lx [i] < weight [i] [j])
                lx [i] = weight [i] [j];
    }
    memset (match, -1, sizeof (match));
    for (int u = 0; u < n; u ++)
        while (1)
        {
            memset (sx, 0, sizeof (sx));
            memset (sy, 0, sizeof (sy));
            if (path (u)) break;
            // 修改标号
            int dx = 0x7FFFFFFF;
            for (i = 0; i < n; i ++)
                if (sx [i])
                    for (j = 0; j < n; j ++)
                        if(!sy [j]) dx = min (lx[i] + ly [j] - weight [i] [j],
dx);
            for (i = 0; i < n; i ++)
            {
                if (sx [i]) lx [i] -= dx;
                if (sy [i]) ly [i] += dx;
            }
        }
        int sum = 0;
        for (i = 0; i < n; i ++)
            sum += weight [match [i]] [i];
```

```
        if (!maxsum)
        {
            sum = -sum;
            for (i = 0; i < n; i ++)
                for (j = 0; j < n; j ++)
                    weight [i] [j] = -weight [i] [j]; // 如果需要保持 weight
[ ] [ ] 原来的值，这里需要将其还原
        }
        return sum;
}
```

3.4 连通性

强连通缩点&双连通缩点

定义：有向图强连通分量在有向图 G 中，如果两个顶点 vi,vj 间（vi<>vj）有一条从 vi 到 vj 的有向路径，同时还有一条从 vj 到 vi 的有向路径，则称两个顶点强连通(strongly connected)。如果有向图 G 的每两个顶点都强连通，称 G 是一个强连通图。非强连通图有向图的极大强连通子图，称为强连通分量(strongly connected components)。

定义：在无向连通图中，如果删除该图的任何一个结点都不能改变该图的连通性，则该图为双连通的无向图。一个连通的无向图是双连通的，当且仅当它没有关结点。

算法：采用 tarjan 算法，用 dfn 和 low 数组进行 dfs 顺序记录，当 low<dfn 时说明能到达前面的点，可以和前面的点构成一个连通分量，只要找到 low==dfn 那个点即可，然后出栈到那个点即可。

双连通分量：双连通分量中删除一个点或删除一条边，连通分量一样连通，是用于求割点或割边的。

**一个顶点 u 是割点，当且仅当满足(1)或(2)

(1) u 为树根，且 u 有多于一个子树。

(2) u 不为树根，且满足存在(u,v)为树枝边(或称父子边，即 u 为 v 在搜索树中的父亲)，使得 DFS(u)<=Low(v)。

**一条无向边(u,v)是桥，当且仅当(u,v)为树枝边，且满足 DFS(u)<Low(v)。

**仙人掌图 if(vd[v]&&dfn[v]!=low[v])return false.不是一个强连分量的图也不是。

强连通缩点

这题是求加多少条边能使原图变成强连通图，先强连通缩点，然后计算入度为 0 和出度为 0 点，取两者中最大的就是答案，因为可以将连通分量变成一个环。

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

```
using namespace std;
int n,m,head[20005],pos,ans,cnt,dfn[20005],low[20005],bch,top,side[2000
5],in[20005],out[20005],stack[20005];
bool vd[20005];
struct egde
{
    int v,next;
}eg[50005];
void insert(int x,int y)
{
    eg[pos].v=y,eg[pos].next=head[x],head[x]=pos++;
}
void tarjan(int s)
{
    vd[s]=1;stack[top++]=s;
    dfn[s]=low[s]=cnt++;
    for(int i=head[s];i>=0;i=eg[i].next)
    {
        if(!vd[eg[i].v])
            tarjan(eg[i].v),low[s]=min(low[s],low[eg[i].v]);
        else if(side[eg[i].v]==-1)
            low[s]=min(low[s],dfn[eg[i].v]);
    }
    if(dfn[s]==low[s])
    {
        int v;
        while(1)
        {
            v=stack[--top];
            side[v]=bch;
            if(v==s)
            {
                bch++;
                break;
            }
        }
    }
}
int main()
{
    int x,y,T;
    scanf("%d",&T);
    while(T--)
    {
```

```
        scanf("%d%d",&n,&m);
        memset(head,-1,sizeof(head));
        pos=0;
        for(int i=0;i<m;i++)
        {
            scanf("%d%d",&x,&y);
            insert(x,y);
        }
        memset(vd,0,sizeof(vd));
        memset(side,-1,sizeof(side));
        ans=0;cnt=0;top=bch=0;
        for(int i=1;i<=n;i++)
            if(!vd[i])
              tarjan(i);
        memset(in,0,sizeof(in));
        memset(out,0,sizeof(out));
        for(int i=1;i<=n;i++)
        {
            for(int j=head[i];j>=0;j=eg[j].next)
            {
                int a=side[i],b=side[eg[j].v];
                if(a!=b)
                {
                    out[a]++,in[b]++;
                }
            }
        }
        int sum1,sum2;
        sum1=sum2=0;
        for(int i=0;i<bch;i++)
        {
            if(in[i]==0)sum1++;
            if(out[i]==0)sum2++;
        }
        ans=max(sum1,sum2);
        if(bch==1)ans=0;
        printf("%d\n",ans);
    }
    return 0;
}
```

双连通缩点

这题是求切开一条边分成两个连通分量，这两个连通分量的权值差最小。双连通缩点，然

后 dfs 计算子数权值，从而计算差值。注意重边，a-b,a-b 说明 a-b 是一个连通分量，要计算重边。

```
#include<cstdio>
#include<string>
#include<cstring>
#include<algorithm>
#include<iostream>
#include<cmath>
#include<vector>
using namespace std;
int n,m,pos,ans,cnt,res[10005],head[10005],col[10005],dfn[10005],low[100
05],bch,ord,stack[10005],top,tr[10005],sum[10005];
vector< pair<int,int> >brige;
bool vd[10005];
struct edge
{
    int v,next;
}eg[40005];
void insert(int x,int y)
{
    eg[pos].v=y,eg[pos].next=head[x],head[x]=pos++;
}
void tarjan(int u,int f)
{
    dfn[u]=low[u]=ord++;
    vd[u]=true;
    stack[top++]=u;
    int flag=0;
    for(int i=head[u];i>=0;i=eg[i].next)
    {
        int v=eg[i].v;
        if(v==f&&!flag&&(flag=1,1))//处理重边
            continue;
        if(!vd[v])
            tarjan(v,u),low[u]=min(low[u],low[v]);
        else if(col[v]==-1)
            low[u]=min(low[u],dfn[v]);
        if(dfn[u]<low[v])//割边条件
            brige.push_back(make_pair(u,v));
    }
    if(low[u]==dfn[u])
    {
        while(stack[top-1]!=u)
            col[stack[--top]]=bch,sum[bch]+=res[stack[top]];
```

```
        top--,sum[bch]+=res[stack[top]],col[u]=bch++;
    }
}
void buildT()
{
    memset(tr,-1,sizeof(tr));
    for(int i=0;i<brige.size();i++)
    {
        int u=brige[i].first,v=brige[i].second;
        eg[pos].v=col[v],eg[pos].next=tr[col[u]];
        tr[col[u]]=pos++;
        eg[pos].v=col[u],eg[pos].next=tr[col[v]];
        tr[col[v]]=pos++;
    }
}
int dfs(int u)
{
    int tem=sum[u];
    vd[u]=1;
    for(int i=tr[u];i>=0;i=eg[i].next)
        if(!vd[eg[i].v])
            tem+=dfs(eg[i].v);
    ans=min(ans,abs(cnt-2*tem));
    return tem;
}
int main()
{
    int x,y;
    while(~scanf("%d%d",&n,&m))
    {
        cnt=0;
        for(int i=0;i<n;i++)
            scanf("%d",&res[i]),cnt+=res[i];
        memset(head,-1,sizeof(head));
        pos=0;
        for(int i=0;i<m;i++)
        {
            scanf("%d%d",&x,&y);
            insert(x,y),insert(y,x);
        }
        bch=ord=top=0;
        memset(col,-1,sizeof(col));
        memset(vd,0,sizeof(vd));
        memset(sum,0,sizeof(sum));
```

```
        brige.clear();
                tarjan(0,0);
        if(bch==1)
            puts("impossible");
        else
        {
            buildT();
            ans=1<<29;
            memset(vd,0,sizeof(vd));
            dfs(0);
            printf("%d\n",ans);
        }
    }
    return 0;
}
```

3.5 2-sat

2-sat 箴言：如果 a 与 b 矛盾，则添加单向边(a,b'),(a',b).

若只要求是否矛盾，只需构图，缩点，然后判断同一连通分量里面有没有矛盾即可。（一般用于二分答案判断）。

输出最小字典序的 2-sat 组合

```
int n,m,pos,head[16005],tail[16005],vd[16005],ok;
queue< int >q1,q2,q3;
struct egde
{
        int v,next;
}eg[80005];
void add(int u,int v)
{
        eg[pos].v=v,eg[pos].next=head[u];
        head[u]=pos++;
        eg[pos].v=u,eg[pos].next=tail[v];
        tail[v]=pos++;
}
void dfs1(int u)
{
```

```
        if(vd[u]==1||ok==1)return ;
        if(vd[u]==0)vd[u]=1,q1.push(u);
        else
        {
                ok=1;return;
        }
        if(vd[u^1]==0)vd[u^1]=2,q3.push(u^1);
        else if(vd[u^1]==1)
        {
                ok=1;return ;
        }
        for(int i=head[u];i>=0;i=eg[i].next)
        {
                int v=eg[i].v;
                dfs1(v);
        }
}
void dfs2(int u)
{
        if(ok==1)return ;
        if(vd[u]==2) return ;
        else if(vd[u]==1)
        {
                ok=1;return ;
        }
        else vd[u]=2,q2.push(u);
        if(vd[u^1]==2)
        {
                ok=1;return ;
        }
        for(int i=tail[u];i>=0;i=eg[i].next)
        {
                int v=eg[i].v;
                dfs2(v);
        }
}
void empty()
{
        while(!q1.empty())
        {
                vd[q1.front()]=0;
                q1.pop();
        }
        while(!q3.empty())
```

```
        {
                vd[q3.front()]=0;
                q3.pop();
        }
        while(!q2.empty())
                q2.pop();
}
void deal()
{
        int cnt=0,flag=0;
        while(cnt<2*n)
        {
                empty();
                for(int i=0;i<2*n;i+=2)
                {
                        if(!vd[i])
                        {
                                ok=0;
                                dfs1(i);
                                if(ok==1)
                                {
                                        ok=0;
                                        empty();
                                        dfs1(i^1);
                                        if(ok==1)
                                        {
                                                flag=1;
                                                break;
                                        }
                                        else
                                        {
                                                while(!q3.empty())
                                                {
                                                        dfs2(q3.front
());
                                                     q2.push(q3.front
());
                                                        q3.pop();
                                                }
                                                while(!q1.empty())
                                                        q1.pop(),cnt++;
                                                while(!q2.empty())
                                                        q2.pop(),cnt++;
                                        }
```

```
                                }
                                else
                                {
                                        while(!q3.empty())
                                        {
                                                dfs2(q3.front());
                                                q2.push(q3.front());
                                                q3.pop();
                                        }
                                        while(!q1.empty())
                                                q1.pop(),cnt++;
                                        while(!q2.empty())
                                                q2.pop(),cnt++;
                                }
                        }
                }
                if(flag)break;
        }
        if(flag)puts("NIE");
        else
        {
                for(int i=0;i<2*n;i++)
                {
                        if(vd[i]==1)
                                printf("%d\n",i+1);
                }
        }
}
int main()
{
        int xx,yy,x,y;
        while(~scanf("%d%d",&n,&m))
        {
                memset(head,-1,sizeof(head));
                memset(tail,-1,sizeof(tail));
                pos=0;
                int flag=0;
                for(int i=0;i<m;i++)
                {
                        scanf("%d%d",&x,&y);
                        x--,y--;
                        if(x!=(y^1))
                                add(x,y^1),add(y,x^1);
                        else flag=1;
```

```
                }
                if(flag==0)
                {
                        memset(vd,0,sizeof(vd));
                        deal();
                }
                else puts("NIE");
        }
        return 0;
}
```

3.6 差分约束

①：对于差分不等式，a - b <= c ，建一条 b 到 a 的权值为 c 的边，求的是最短路，得到的是最大值
②：对于不等式 a - b >= c ，建一条 b 到 a 的权值为 c 的边，求的是最长路，得到的是最小值
③：存在负环的话是无解
④：求不出最短路（dist[ ]没有得到更新）的话是任意解
一种建图方法：
设 x[i]是第 i 位置（或时刻）的值（跟所求值的属性一样），那么把 x[i]看成数列，前 n 项和为 s[n]，则 x[i] = s[i] - s[i-1];
注意要点：若有规定方向性，例如右边的一点比左边的大，那么要一直遵守这个规定，构图及 spfa 都要遵循这个方向
如下面这题房子从左到右方向固定所有操作都要按统一的方向，即从小到大。

```
const int size=1005,INF=1<<30;
int n,m,head[size],pos;
bool vd[size];
int d[size],cnt[size];
struct egde
{
        int v,w,next;
}eg[size*10];
struct node
{
        int num,flag;
}nd[size];
void add(int x,int y,int w)
{
        eg[pos].v=y,eg[pos].w=w,eg[pos].next=head[x];
        head[x]=pos++;
}
bool cmp(node a,node b)
```

```
{
        return a.num<b.num;
}
int spfa(int st,int fn)
{
        queue< int > q;
        for(int i=0;i<=n;i++)
                d[i]=INF,vd[i]=0,cnt[i]=0;
        d[st]=0;vd[st]=1;q.push(st);
        while(!q.empty())
        {
                int u=q.front();
                vd[u]=0;q.pop();
                for(int i=head[u];i>=0;i=eg[i].next)
                {
                        int v=eg[i].v,w=eg[i].w;
                        if(d[v]>d[u]+w)
                        {
                                d[v]=d[u]+w;
                                if(!vd[v])
                                {
                                        vd[v]=1;
                                        cnt[v]++;
                                        q.push(v);
                                        if(cnt[v]>n)return -1;
                                }
                        }
                }
        }
        if(d[fn]==INF)return -1;
        return d[fn];
}
int main()
{
        int T;
        scanf("%d",&T);
        for(int t=1;t<=T;t++)
        {
                scanf("%d%d",&n,&m);
                memset(head,-1,sizeof(head));
                pos=0;
                int Min=INF,Max=-INF,tab1,tab2;
                for(int i=1;i<=n;i++)
                {
```

```
                        scanf("%d",&nd[i].num);
                        nd[i].flag=i;
                        if(nd[i].num>Max)
                                Max=nd[i].num,tab1=i;
                        if(nd[i].num<Min)
                                Min=nd[i].num,tab2=i;
                }
                sort(nd+1,nd+n+1,cmp);
                for(int i=1;i<n;i++)
                {
                        int y=max(nd[i].flag,nd[i+1].flag),x=min(nd[i].f
lag,nd[i+1].flag);
                        add(x,y,m);
                }
                for(int i=1;i<n;i++)
                {
                        add(i+1,i,-1);
                }
                int x=min(tab1,tab2),y=max(tab1,tab2);
                printf("Case %d: %d\n",t,spfa(x,y));
        }
        return 0;
}
```

3.7 网络流

EK

```
using namespace std;
const int INF=1<<30;
const int size=205;
int n,m,cap[size][size],flow[size][size],inc[size],per[size];
int EK(int st,int fn)
{
        int ans=0;
        queue< int > q;
        memset(flow,0,sizeof(flow));
        while(1)
        {
                memset(inc,0,sizeof(inc));
                inc[st]=INF;q.push(st);
                while(!q.empty())
                {
```

```
                        int u=q.front();
                        q.pop();
                        for(int v=1;v<=m;v++)
                        {
                                if(!inc[v]&&cap[u][v]>flow[u][v])
                                {
                                inc[v]=min(inc[u],cap[u][v]-flow[u][v]);
                                        per[v]=u;
                                        q.push(v);
                                }
                        }
                }
                if(inc[fn]==0)
                        break;
                for(int u=fn;u!=st;u=per[u])
                {
                        flow[per[u]][u]+=inc[fn];
                        flow[u][per[u]]-=inc[fn];
                }
                ans+=inc[fn];
        }
        return ans;
}
int main()
{
        int x,y,w;
        while(~scanf("%d%d",&n,&m))
        {
                memset(cap,0,sizeof(cap));
                for(int i=0;i<n;i++)
                {
                        scanf("%d%d%d",&x,&y,&w);
                        cap[x][y]+=w;//构图是时注意可能有重的，要加
                }
                printf("%d\n",EK(1,m));
        }
        return 0;
}
```

ISAP

```
const int MAXN = 100000 + 10;
const int INF = 1<<30;
const double eps = 1e-8;
```

```
struct EDGE {
    int v,c,next;
}E[MAXN*4];

int G[205][205];
int head[MAXN],cur[MAXN],pre[MAXN],cnt[MAXN],dis[MAXN];
int n,m,s,t,tot,low,N;

void add(int u,int v,int c) {
    E[tot].v=v;
    E[tot].c=c;
    E[tot].next=head[u];
    head[u]=tot++;
}

void addedge(int u,int v,int c) {
    add(u,v,c);
    add(v,u,0);
}

void rebfs() {    //反向 bfs 求最短距离
    for(int i=0;i<N;i++) cnt[i]=0,dis[i]=N;
    queue<int> q;
    dis[t]=0;
    cnt[0]=1;
    q.push(t);
    while(!q.empty()) {
        int u=q.front();
        q.pop();
        for(int e=head[u];e!=-1;e=E[e].next) {
            if(E[e^1].c==0||dis[E[e].v]<N) continue;
            dis[E[e].v]=dis[u]+1;
            cnt[dis[E[e].v]]++;
            q.push(E[e].v);
        }
    }
}

int ISAP() {
    rebfs();
    for(int i=0;i<N;i++) cur[i]=head[i];
    int flow=0,u=s,v,e;
    low=INF;
    while(dis[s]<N) {
```

```
        for(e=cur[u];e!=-1;e=E[e].next)
            if(E[e].c&&dis[u]==dis[v=E[e].v]+1) break;
        if(e!=-1) {
            cur[u]=e; pre[v]=e; low=min(low,E[e].c);
            u=v;
            if(u==t) {
                for(;u!=s;u=E[pre[u]^1].v)
                {
                    E[pre[u]].c-=low;
                    E[pre[u]^1].c+=low;
                }
                flow+=low;
                low=INF;
            }
        }
        else {
            if(--cnt[dis[u]]==0) break;//GAP 优化
            dis[u]=N; cur[u]=head[u];
            for(e=head[u];e!=-1;e=E[e].next)
                if(E[e].c>0&&dis[u]>dis[E[e].v]+1)
                    dis[u]=dis[E[e].v]+1;
            cnt[dis[u]]++;
            if(u!=s) u=E[pre[u]^1].v;
        }
    }
    return flow;
}
```

数据结构

4.1 树状数组

树状数组

```
int bit(int n)
{
        return n&(-n);
```

```
}
void insert(int n,int k)
{
        while(n<100005)
        {
                c[n]+=k;
                n+=bit(n);
        }
}
int sum(int n)
{
        int ans=0;
        while(n)
        {
                ans+=c[n];
                n-=bit(n);
        }
        return ans;
}
int find(int n)//求树状数组中的第 k 大的数
{
        int cnt=0,num=0;
        for(int i=20;i>=0;i--)
        {
                num+=(1<<i);
                if(num>100000||cnt+c[num]>=n)num-=(1<<i);
                else cnt+=c[num];
        }
        return num+1;
}
```

4.2 线段树

```
#define LG __int64    //区间乘积
using namespace std;
const int N=50005,mod=1000000007;
int n,m;
LG a[N];
struct tree
{
    int l,r;
```

```
    LG mul;
}T[N*4];
void Mod(LG &k)
{
    if(k>=mod)
        k=k%mod;
}
void buildT(int l,int r,int p)
{
    T[p].l=l,T[p].r=r;
    if(l==r)
    {
        T[p].mul=a[l];Mod(T[p].mul);
    }
    else
    {
        int mid=(l+r)>>1;
        buildT(l,mid,p<<1);buildT(mid+1,r,(p<<1)+1);
        T[p].mul=T[p<<1].mul*T[(p<<1)+1].mul;
        Mod(T[p].mul);
    }
}
void update(int p,int tab,int num)
{
    if(T[p].l==T[p].r)
    {
        T[p].mul=num;
        return ;
    }
    int mid=(T[p].l+T[p].r)>>1;
    if(tab>mid)
    {
        update((p<<1)+1,tab,num);
    }
    else
    {
        update(p<<1,tab,num);
    }
    T[p].mul=T[p<<1].mul*T[(p<<1)+1].mul;
    Mod(T[p].mul);
}
LG find(int l,int r,int p)
{
    if(T[p].l==l&&T[p].r==r)
```

```
    {
        return T[p].mul;
    }
    else
    {
        int mid=(T[p].l+T[p].r)>>1;
        LG ans;
        if(l>mid)
            ans=find(l,r,(p<<1)+1);
        else if(r<=mid)
            ans=find(l,r,(p<<1));
        else
        {
            ans=find(l,mid,(p<<1))*find(mid+1,r,(p<<1)+1);
            Mod(ans);
        }
        return ans;
    }
}
int main()
{
    int T,x,y,op;
    scanf("%d",&T);
    while(T--)
    {
        scanf("%d",&n);
        for(int i=1;i<=n;i++)
            scanf("%I64d",&a[i]);
        buildT(1,n,1);
        scanf("%d",&m);
        for(int i=0;i<m;i++)
        {
            scanf("%d%d%d",&op,&x,&y);
            if(op==0)
            {
                if(x>y)swap(x,y);
                printf("%I64d\n",find(x,y,1));
            }
            else
            {
                update(1,x,y);
            }
        }
    }
```

```
    return  0;
}
```

4.3 后缀数组

模板（倍增算法）

```
#define ffor(i,a,n) for(int i=a;i<n;++i)
#define rep(i,n) ffor(i,0,n)
using namespace std;
#define maxn 200002
int wa[maxn],wb[maxn],wv[maxn],wc[maxn];
int cmp(int *r,int a,int b,int l)
{
    return r[a]==r[b]&&r[a+l]==r[b+l];
}
void da(int *r,int *sa,int n,int m)
{
    int *x=wa,*y=wb,*t;
    rep(i,m) wc[i]=0;
    rep(i,n) wc[x[i]=r[i]]++;
    ffor(i,1,m) wc[i]+=wc[i-1];
    for(int i=n-1;i>=0;i--) sa[--wc[x[i]]]=i;
    for(int j=1,p=1;p<n;j*=2,m=p)
    {
        p=0;
        for(int i=n-j;i<n;i++) y[p++]=i;
        rep(i,n) if(sa[i]>=j) y[p++]=sa[i]-j;
        rep(i,n) wv[i]=x[y[i]];
        rep(i,m) wc[i]=0;
        rep(i,n) wc[wv[i]]++;
        ffor(i,1,m) wc[i]+=wc[i-1];
        for(int i=n-1;i>=0;i--) sa[--wc[wv[i]]]=y[i];
        swap(x,y);
        p=1,x[sa[0]]=0;
        for(int i=1;i<n;i++)
            x[sa[i]] = cmp( y, sa[i-1], sa[i], j)?p-1:p++;
    }
    return;
}
int rk[maxn],height[maxn];
void calheight(int *r,int *sa,int n)
{
    int i,j,k=0;
```

```
    for(i=1;i<=n;i++) rk[sa[i]]=i;
    for(i=0;i<n;height[rk[i++]]=k)
        for(k?k--:0,j=sa[rk[i]-1];r[i+k]==r[j+k];k++);
    return;
}

char s[maxn];
int r[maxn],sa[maxn];
int main()//求最长公共子串
{

    while(~scanf("%s",s))
    {
        int i,j,n,ans=0;
        j=strlen(s);
        s[j]=1;
        scanf("%s",s+j+1);
        n=strlen(s);
        for(i=0;i<n;i++) r[i]=s[i];
        r[n]=0;
        da(r,sa,n+1,128);
        calheight(r,sa,n);
        for(i=2;i<=n;i++)
            if(height[i]>ans)
                if((j<sa[i-1] && j>sa[i])|| (j>sa[i-1] && j<sa[i]))
ans=height[i];
        printf("%d\n",ans);
    }
    return 0;
}
```

模板（DC3 算法）

```
#define ffor(i,a,n) for(int i=a;i<n;++i)
#define rep(i,n) ffor(i,0,n)
using namespace std;
#define maxn 200004
#define F(x) ((x)/3+((x)%3==1?0:tb))
#define G(x) ((x)<tb?(x)*3+1:((x)-tb)*3+2)
int wa[maxn],wb[maxn],wv[maxn],wc[maxn];
int c0(int *r,int a,int b)
{
    return r[a]==r[b]&&r[a+1]==r[b+1]&&r[a+2]==r[b+2];
}
```

```
int c12(int k,int *r,int a,int b)
{
    if(k==2) return r[a]<r[b]||r[a]==r[b]&&c12(1,r,a+1,b+1);
    else return r[a]<r[b]||r[a]==r[b]&&wv[a+1]<wv[b+1];
}
void sort(int *r,int *a,int *b,int n,int m)
{
    rep(i,n) wv[i]=r[a[i]];
    rep(i,m) wc[i]=0;
    rep(i,n) wc[wv[i]]++;
    ffor(i,1,m) wc[i]+=wc[i-1];
    for(int i=n-1;i>=0;i--) b[--wc[wv[i]]]=a[i];
    return;
}
void dc3(int *r,int *sa,int n,int m)
{
    int i,j,*rn=r+n,*san=sa+n,ta=0,tb=(n+1)/3,tbc=0,p;
    r[n]=r[n+1]=0;
    for(i=0;i<n;i++) if(i%3!=0) wa[tbc++]=i;
    sort(r+2,wa,wb,tbc,m);
    sort(r+1,wb,wa,tbc,m);
    sort(r,wa,wb,tbc,m);
    for(p=1,rn[F(wb[0])]=0,i=1;i<tbc;i++)
        rn[F(wb[i])]=c0(r,wb[i-1],wb[i])?p-1:p++;
    if(p<tbc) dc3(rn,san,tbc,p);
    else for(i=0;i<tbc;i++) san[rn[i]]=i;
    for(i=0;i<tbc;i++) if(san[i]<tb) wb[ta++]=san[i]*3;
    if(n%3==1) wb[ta++]=n-1;
    sort(r,wb,wa,ta,m);
    for(i=0;i<tbc;i++) wv[wb[i]=G(san[i])]=i;
    for(i=0,j=0,p=0;i<ta && j<tbc;p++)
        sa[p]=c12(wb[j]%3,r,wa[i],wb[j])?wa[i++]:wb[j++];
    for(;i<ta;p++) sa[p]=wa[i++];
    for(;j<tbc;p++) sa[p]=wb[j++];
    return;
}
int rk[maxn],height[maxn];
void calheight(int *r,int *sa,int n)
{
    int i,j,k=0;
    for(i=1;i<=n;i++) rk[sa[i]]=i;
    for(i=0;i<n;height[rk[i++]]=k)
        for(k?k--:0,j=sa[rk[i]-1];r[i+k]==r[j+k];k++);
    return;
```

```
}

char s[maxn];
int r[maxn*3],sa[maxn*3];
int main()//求最长公共子串
{
    while(~scanf("%s",s))
    {
        int i,j,n,ans=0;
        j=strlen(s);
        s[j]=1;
        scanf("%s",s+j+1);
        n=strlen(s);
        for(i=0;i<n;i++) r[i]=s[i];
        r[n]=0;
        dc3(r,sa,n+1,128);
        calheight(r,sa,n);
        for(i=2;i<=n;i++)
            if(height[i]>ans)
                if((j<sa[i-1] && j>sa[i])|| (j>sa[i-1] && j<sa[i]))
ans=height[i];
        printf("%d\n",ans);
    }
    return 0;
}
```

4.4 trie 树

Trie

```
struct node
{
        int next[30];
        int cnt;//结尾标记
}nd[500005];
int pos=0;
void clean(int n)//清空节点，方便再利用
{
        memset(nd[n].next,-1,sizeof(nd[n].next));
        nd[n].cnt=0;
}
void creat(char s[],node &T)//T 记得要初始化
```

```
{
        int len=strlen(s),i=0;
        node *p=&T;
        while(i<len)
        {
                int tem=s[i]-'a';
                if(p->next[tem]==-1)
                {
                        clean(pos);p->next[tem]=pos++;
                }
                nd[p->next[tem]].cnt++;
                p=&nd[p->next[tem]];
                i++;
        }
}
int find(char s[],node &T)
{
        int len=strlen(s),i=0;
        node *p=&T;
        while(i<len)
        {
                int tem=s[i]-'a';
                if(p->next[tem]==-1)
                        return 0;
                else
                        p=&nd[p->next[tem]];
                i++;
        }
        return p->cnt;
}
```

4.6 AC 自动机

```
#include <iostream>
using namespace std;
const int kind = 26;
struct node{
    node *fail;       //失败指针
    node *next[kind]; //Tire 每个节点的 26 个子节点（最多 26 个字母）
    int count;        //是否为该单词的最后一个节点
    node(){           //构造函数初始化
       fail=NULL;
       count=0;
```

```
        memset(next,NULL,sizeof(next));
    }
}*q[500001];          //队列，方便用于 bfs 构造 fail 指针
char keyword[51];     //输入的单词
char str[1000001];    //模式串
int head,tail;        //队列的头尾指针

void insert(char *str,node *root){     //建字典树
    node *p=root;
    int i=0,index;
    while(str[i]){
        index=str[i]-'a';
        if(p->next[index]==NULL) p->next[index]=new node();
        p=p->next[index];
        i++;
    }
    p->count++;           //每个单词的末尾字母标记 count 为 1，代表一个单词
}
/*
在字典树上构造 fail 指针。构造失败指针的过程概括起来就一句话：
设这个节点上的字母为 C，沿着他父亲的失败指针走，直到走到一个节点，
他的儿子中也有字母为 C 的节点。然后把当前节点的失败指针指向那个字母也为 C 的儿子。
如果一直走到了根节点都没找到，那就把失败指针指向根节点。

所以构造 fail 指针 需要用到 BFS。 保证是按层遍历字典树。
*/
void build_ac_automation(node *root){      //构建 fail 指针
    int i;
    root->fail=NULL;       //根节点 fail 指针指向空值
    q[head++]=root;        //根节点入队
    while(head!=tail){
        node *temp=q[tail++];
        node *p=NULL;
        for(i=0;i<26;i++){
            if(temp->next[i]!=NULL){
                if(temp==root) temp->next[i]->fail=root;
                //根节点每个儿子的 fail 指针为根节点
                else{
                    p=temp->fail;
                    while(p!=NULL){    //P 不为空，即未到达根节点
                        if(p->next[i]!=NULL){  //找到了包含当前字母儿子的父
节点
                            temp->next[i]->fail=p->next[i];//将子结点的 fail
指针指向此节点
```

```
                        break;
                    }
                    p=p->fail;     //未找到，则到其 fail 指针节点处继续找
                }
                if(p==NULL) temp->next[i]->fail=root;
            }
            q[head++]=temp->next[i];   //子节点入队
        }
    }
}
/*
匹配过程分两种情况：
(1)当前字符匹配，表示从当前节点沿着树边有一条路径可以到达目标字符，
此时只需沿该路径走向下一个节点继续匹配即可，目标字符串指针移向下个字符继续匹配；
(2)当前字符不匹配，则去当前节点失败指针所指向的字符继续匹配，
匹配过程随着指针指向 root 结束。重复这 2 个过程中的任意一个，直到模式串走到结尾为止。
*/
int query(node *root){
    int i=0,cnt=0,index,len=strlen(str);
    node *p=root;
    while(str[i]){
        index=str[i]-'a';
        while(p->next[index]==NULL && p!=root) p=p->fail; //当在字典树上找不到 c 字符，那么就根据 fail 指针回退
       //直到找到 或者到达根节点
        p=p->next[index];
        p=(p==NULL)?root:p;
        node *temp=p;
        while(temp!=root && temp->count!=-1){  //根据 fail 指针回退，直到根节点
            cnt+=temp->count;
            temp->count=-1;        //避免重复记录
            temp=temp->fail;
        }
        i++;
    }
    return cnt;
}
int main(){
    int n,t;
    scanf("%d",&t);
    while(t--){
```

```
        head=tail=0;
        node *root=new node();
        scanf("%d",&n);
        getchar();
        while(n--){
            gets(keyword);
            insert(keyword,root);
        }
        build_ac_automation(root);
        scanf("%s",str);
        printf("%d\n",query(root));
    }
    return 0;
}
```

4.7 公共祖先和区间极值查询

Tarjan(O(N+M))

```
void tarjan(int n,int dth,int f)
{
        vd[n]=1;dis[n]=dth;tr[n]=f;//是否属于同一个连通分支标志
        for(int i=0;i<q[n].size();i++)
        {
                int v=q[n][i].v,id=q[n][i].w;//问题的标号
                if(vd[v])
                {
                        if(tr[n]==tr[v])
                        ans[id]=dis[n]+dis[v]-2*dis[find(v)];
                            //和并查集的 find 一致,find(v)是公共祖先
                }
        }
        for(int i=0;i<eg[n].size();i++)
        {
                int v=eg[n][i].v,w=eg[n][i].w;
                if(!vd[v])
                {
                        tarjan(v,dth+w,f);
                        fa[v]=n;回溯关键
                }
        }
}
```

ST 倍增算法(O(nlong(n)))

```
void ST(int n)//dp[i][j]表示 i~i+(1<<j)-1 中的最值
{
        for(int i=1;i<=n;i++)
                dp[i][0]=a[i];
        for(int j=1;(1<<j)<=n;j++)
                for(int i=1;i+(1<<j)-1<=n;i++)
                {
                        dp[i][j]=max(dp[i][j-1],dp[i+(1<<(j-1))][j-1]);
                }
}
int getmax(int a,int b)
{
        int k=(log(b-a+1.0)/log(2.0));
        return max(dp[a][k],dp[b-(1<<k)+1][k]);
}
```

4.8 Dacing Link

精确覆盖

```
const int MAXR = 1000 + 10;
const int MAXC = 100 + 10;
const int INF = 1<<30;
const double eps = 1e-8;

int
L[MAXR*MAXC],R[MAXR*MAXC],U[MAXR*MAXC],D[MAXR*MAXC],C[MAXR*MAXC],ROW[MA
XR*MAXC];
int H[MAXR],S[MAXR],ans[MAXR],cnt,sz,head,n,m;

void init() {
    memset(H,-1,sizeof(H));
    head=0;
    for(sz=0;sz<=m;sz++) {
        S[sz]=0;
        D[sz]=U[sz]=sz;
        L[sz+1]=sz;
        R[sz]=sz+1;
    }
    R[sz-1]=head;
    L[head]=sz-1;
```

```
}

void link(int r,int c) {
    S[C[sz]=c]++;
    D[sz]=D[c];
    U[D[c]]=sz;
    U[sz]=c;
    D[c]=sz;
    if(H[r]<0) H[r]=R[sz]=L[sz]=sz;
    else {
        R[sz]=R[H[r]];
        L[R[sz]]=sz;
        L[sz]=H[r];
        R[H[r]]=sz;
    }
    ROW[sz++]=r;
}

void Remove(int c) {
    L[R[c]]=L[c];
    R[L[c]]=R[c];
    for(int i=D[c];i!=c;i=D[i])
        for(int j=R[i];j!=i;j=R[j]) {
            U[D[j]]=U[j];
            D[U[j]]=D[j];
            --S[C[j]];
        }
}

void Resume(int c) {
    L[R[c]]=R[L[c]]=c;
    for(int i=U[c];i!=c;i=U[i])
        for(int j=L[i];j!=i;j=L[j]) {
            U[D[j]]=D[U[j]]=j;
            S[C[j]]++;
        }
}

bool DLX(int k) {
    if(R[head]==head) {
        cnt=k;
        return true;
    }
    int s=INF,c;
```

```
    for(int i=R[head];i!=head;i=R[i])
        if(S[i]<s)
            s=S[c=i];
    Remove(c);
    for(int i=D[c];i!=c;i=D[i]) {
        ans[k]=ROW[i];
        for(int j=R[i];j!=i;j=R[j]) Remove(C[j]);
        if(DLX(k+1)) return true;
        for(int j=L[i];j!=i;j=L[j]) Resume(C[j]);
    }
    Resume(c);
    return false;
}

int main() {
    while(scanf("%d%d",&n,&m)!=EOF) {
        init();
        int tm,t;
        for(int i=1;i<=n;i++) {
            scanf("%d",&tm);
            for(int j=0;j<tm;j++)
            {
                scanf("%d",&t);
                link(i,t);
            }
        }
        if(DLX(0)) {
            printf("%d ",cnt);
            for(int i=0;i<cnt;i++)
                printf("%d ",ans[i]);
            printf("\n");
        }
        else printf("NO\n");
    }
    return 0;
}
```

重复覆盖

```
const int MAXR = 100 + 10;
const int MAXC = 100 + 10;
```

```
const int INF = 1<<30;
const double eps = 1e-8;

int
L[MAXR*MAXC],R[MAXR*MAXC],U[MAXR*MAXC],D[MAXR*MAXC],C[MAXR*MAXC],ROW[MA
XR*MAXC];
int H[MAXR],S[MAXR],ans,sz,head,n;
double x[MAXR],y[MAXR],r[MAXR];

void init() {
    ans=INF;
    memset(H,-1,sizeof(H));
    head=0;
    for(sz=0;sz<=n;sz++) {
        S[sz]=0;
        D[sz]=U[sz]=sz;
        L[sz+1]=sz;
        R[sz]=sz+1;
    }
    R[sz-1]=head;
    L[head]=sz-1;
}

void link(int r,int c) {
    S[C[sz]=c]++;
    D[sz]=D[c];
    U[D[c]]=sz;
    U[sz]=c;
    D[c]=sz;
    if(H[r]<0) H[r]=R[sz]=L[sz]=sz;
    else {
        R[sz]=R[H[r]];
        L[R[sz]]=sz;
        L[sz]=H[r];
        R[H[r]]=sz;
    }
    ROW[sz++]=r;
}

int h()  //启发函数
{
    int cnt=0;
    bool Hash[MAXC]={0};
    for (int c=R[head];c!=head;c=R[c])
```

```
        if (!Hash[c])
        {
            Hash[c]=true;
            cnt++;
            for (int i=D[c];i!=c;i=D[i])
                for (int j=R[i];j!=i;j=R[j])
                    Hash[C[j]]=true;
        }
    return cnt;
}

void Remove(int c) {
    for(int i=D[c];i!=c;i=D[i]){
        L[R[i]]=L[i];
        R[L[i]]=R[i];
        S[C[i]]--;
    }
}

void Resume(int c) {
    for(int i=U[c];i!=c;i=U[i]){
        L[R[i]]=R[L[i]]=i;
        S[C[i]]++;
    }
}

void DLX(int k) {
    if(k+h()>=ans) return;
    if(R[head]==head){
        ans=k;
        return;
    }
    int s=INF,c;
    for(int i=R[head];i!=head;i=R[i])
        if(S[i]<s)  s=S[c=i];
    for(int i=U[c];i!=c;i=U[i]) {
        Remove(i);
        for(int j=R[i];j!=i;j=R[j]) Remove(j);
        DLX(k+1);
        for(int j=L[i];j!=i;j=L[j]) Resume(j);
        Resume(i);
    }
}
```

```
bool incircle(int i,int j) {
    return
(x[i]-x[j])*(x[i]-x[j])+(y[i]-y[j])*(y[i]-y[j])<r[i]*r[i]+eps;
}

int main() {
    while(scanf("%d",&n)!=EOF){
        init();
        for(int i=1;i<=n;i++)
            scanf("%lf%lf%lf",&x[i],&y[i],&r[i]);
        for(int i=1;i<=n;i++)
            for(int j=1;j<=n;j++)
                if(incircle(i,j))
                    link(i,j);
        DLX(0);
        printf("%d\n",ans);
    }
    return 0;
}
```

计算几何

5.1 各种公式

(1).注意 **-0.00.**

(2).涉及乘法的时候 注意保证不越界 而且要保证精度

(3).[Pick 定理**]** 设以整数点为顶点的多边形的面积为 **S**， 多边形内部的整数点数为 **N**，多边形边界上的整数点数为 **L**， 则 **N + L/2 - 1 = S**

(4) 费马点 对于三角形：当三个角都小于 **120** 度，则费马点在三角形内部与任意两定点的连线构成的角都为 **120** 度； 若存在一个角大于等于 **120** 度，则费马点为此角顶点。 对于平面四边形：若为凸四边形，则费马点为两对角线交点；对于凹四边形，为其凹顶点。 在不是凸四边形的情况下，只要枚举四个顶点即可。

(5)计算四面体体积，欧拉四面体公式

```
a=OA, b=OB, c=OC, l=AB, m=BC, n=CA
double v_4mianti(double a,double b,double c,double l,double n,double m)
{
    return sqrt(4*a*a*b*b*c*c-a*a*(b*b+c*c-m*m)*(b*b+c*c-m*m)-
```

```
        b*b*(c*c+a*a-n*n)*(c*c+a*a-n*n)-c*c*(a*a+b*b-l*l)*(a*a+b*b-l*l)
        +(a*a+b*b-l*l)*(b*b+c*c-m*m)*(c*c+a*a-n*n))/12;
}
```

(6).三角形

1. 半周长 P=(a+b+c)/2
2. 面积 S=aHa/2=absin(C)/2=sqrt(P(P-a)(P-b)(P-c))
3. 中线 Ma=sqrt(2(b^2+c^2)-a^2)/2=sqrt(b^2+c^2+2bccos(A))/2
4. 角平分线 Ta=sqrt(bc((b+c)^2-a^2))/(b+c)=2bccos(A/2)/(b+c)
5. 高线 Ha=bsin(C)=csin(B)=sqrt(b^2-((a^2+b^2-c^2)/(2a))^2)
6. 内切圆半径 r=S/P=asin(B/2)sin(C/2)/sin((B+C)/2)
=4Rsin(A/2)sin(B/2)sin(C/2)=sqrt((P-a)(P-b)(P-c)/P)
=Ptan(A/2)tan(B/2)tan(C/2)
7. 外接圆半径 R=abc/(4S)=a/(2sin(A))=b/(2sin(B))=c/(2sin(C))

(7).四边形

D1,D2 为对角线,M 对角线中点连线,A 为对角线夹角

1. a^2+b^2+c^2+d^2=D1^2+D2^2+4M^2
2. S=D1D2sin(A)/2 (以下对圆的内接四边形)
3. ac+bd=D1D2
4. S=sqrt((P-a)(P-b)(P-c)(P-d)),P 为半周长

(8).正 n 边形

R 为外接圆半径,r 为内切圆半径

1. 中心角 A=2PI/n
2. 内角 C=(n-2)PI/n
3. 边长 a=2sqrt(R^2-r^2)=2Rsin(A/2)=2rtan(A/2)
4. 面积 S=nar/2=nr^2tan(A/2)=nR^2sin(A)/2=na^2/(4tan(A/2))

(9).圆

1. 弧长 l=rA
2. 弦长 a=2sqrt(2hr-h^2)=2rsin(A/2)
3. 弓形高 h=r-sqrt(r^2-a^2/4)=r(1-cos(A/2))=atan(A/4)/2
4. 扇形面积 S1=rl/2=r^2A/2
5. 弓形面积 S2=(rl-a(r-h))/2=r^2(A-sin(A))/2

(10).棱柱

1. 体积 V=Ah,A 为底面积,h 为高
2. 侧面积 S=lp,l 为棱长,p 为直截面周长

3．全面积 T=S+2A

(11).棱锥

1．体积 V=Ah/3,A 为底面积,h 为高 (以下对正棱锥)
2．侧面积 S=lp/2,l 为斜高,p 为底面周长
3．全面积 T=S+A

(12).棱台

1．体积 V=(A1+A2+sqrt(A1A2))h/3,A1.A2 为上下底面积,h 为高 (以下为正棱台)
2．侧面积 S=(p1+p2)l/2,p1.p2 为上下底面周长,l 为斜高
3．全面积 T=S+A1+A2

(13).圆柱

1．侧面积 S=2PIrh
2．全面积 T=2PIr(h+r)
3．体积 V=PIr^2h

(14).圆锥

1．母线 l=sqrt(h^2+r^2)
2．侧面积 S=PIrl
3．全面积 T=PIr(l+r)
4．体积 V=PIr^2h/3

(15).圆台

1．母线 l=sqrt(h^2+(r1-r2)^2)
2．侧面积 S=PI(r1+r2)l
3．全面积 T=PIr1(l+r1)+PIr2(l+r2)
4．体积 V=PI(r1^2+r2^2+r1r2)h/3

(16).球

1．全面积 T=4PIr^2
2．体积 V=4PIr^3/3

(17).球台

1. 侧面积 S=2PIrh
2. 全面积 T=PI(2rh+r1^2+r2^2)
3. 体积 V=PIh(3(r1^2+r2^2)+h^2)/6

(18).球扇形

球扇形:
1. 全面积 T=PIr(2h+r0),h 为球冠高,r0 为球冠底面半径
2. 体积 V=2PIr^2h/3

(19).圆柱正交体积(2*r<=h)

1.两个圆柱：16/3*r^3
2. 三个圆柱:（16-8*sqrt（2））*r^3

5.2 基本类型定义

```
//二维
struct point{ double x,y;};        //点
struct beeline{    double A,B,C;};    //直线 (直线方程)
struct line{ point a,b;};        //直线 （两点式）
struct segment{ point a,b;};    // 线段

//三维
struct point3 {
    double x, y, z;
    point3(){};
    point3(double xx, double yy, double zz) : x(xx), y(yy), z(zz) {}
};
struct plane{ point3 a, b, c;};
```

5.3 基础函数

```
// 大小比较
const double eps = 1e-6;
bool dy(double x,double y)    {    return x > y + eps;}    // x > y
bool xy(double x,double y)    {    return x < y - eps;}    // x < y
bool dyd(double x,double y)    {     return x > y - eps;}    // x >= y
bool xyd(double x,double y)    {    return x < y + eps;}     // x <= y
bool dd(double x,double y)     {    return fabs( x - y ) < eps;}  // x ==
y
double disp2p(point a,point b) //  a b 两点之间的距离
{
    return sqrt( ( a.x - b.x ) * ( a.x - b.x ) + ( a.y - b.y ) * ( a.y - b.y ) );
}
// 叉积 (三点)
double crossProduct(point a,point b,point c)//向量 ac 在 ab 的方向 顺时针是
正
{
    return (c.x - a.x)*(b.y - a.y) - (b.x - a.x)*(c.y - a.y);
}
//叉积（三点坐标）
double crossProduct(double x0,double y0,double x1,double y1,double
x2,double y2)
{
    return (x2 - x0)*(y1 - y0) - (x1 - x0)*(y2 - x0);
}
```

5.4 各种极角排序

```
/*************************************************************************
极角排序，以 C 为顶点，按象限极角排序，相同角度，距离近的排前面
*************************************************************************/
int quad(point a)// 判断象限的函数，每个象限包括半个坐标轴
{
    if( dy(a.x,0) && xyd(a.y,0) ) return 1;
    if( xyd(a.x,0) && dy(a.y,0) ) return 2;
    if( xy(a.x,0) && xyd(a.y,0) ) return 3;
    if( dyd(a.x,0) && xy(a.y,0) ) return 4;
    return 0;        // 和原点重合 ，一般不会有这种数据的。。
}
bool cmp(point& a,point& b)
{
```

```
    point p1 = a,p2 = b;
    p1.x -= C.x; p1.y -= C.y;
    p2.x -= C.x; p2.y -= C.y;
    int l1,l2;
    l1 = quad(p1); l2 = quad(p2);
    if( l1 == l2 )
    {
        double c = crossProduct(C,a,b);
        return xy(c,0) || dd(c,0.0) && xy(fabs(a.x),fabs(b.x));
    }
    return l1 < l2;
}
/*********************************************************************
              极角排序，以 C 为顶点，按照向量排序，范围(-180°, +180°]
*********************************************************************/
bool cmp(point& a,point& b)
{
    double t1 = atan2(a.y - C.y, a.x - C.x);
    double t2 = atan2(b.y - C.y, b.x - C.x);
    if( dd(t1, t2) ) return xy(fabs(a.x),fabs(b.x));
    return xy(a.t, b.t);
}
```

5.5 点、直线、线段

5.5.1 两直线位置关系

```
/*********************************************************************
线的位置关系：     平行 垂直 相交即不平行
*********************************************************************/
//判断直线是否平行，平行返回 1
bool parallel(point u1,point u2,point v1,point v2)
{
    return dd( (u1.x - u2.x)*(v1.y - v2.y) - (v1.x - v2.x)*(u1.y - u2.y) ,
0.0 );
}
bool parallel(line u,line v)
```

```
{
    return dd( (u.a.x - u.b.x)*(v.a.y - v.b.y) - (v.a.x - v.b.x)*(u.a.y -
u.b.y) , 0.0 );
}
//判断两直线是否垂直，垂直返回 1
bool perpendicular(point u1,point u2,point v1,point v2)
{
    return dd((u1.x - u2.x)*(v1.x - v2.x) + (u1.y - u2.y)*(v1.y - v2.y),0.0);
}
```

5.5.2 判断两线段是否相交

```
//判断线段 p1p2 与 p3p4 是否相交，包括端点与端点，端点与线段
const double eps = 1e-10;
struct point{    double x,y;        };
double crossProduct(point a, point b, point c )
bool onSegment(point a, point b, point c)
bool s2s_inst(point p1,point p2, point p3, point p4)
{
    double d1 = crossProduct(p3,p4,p1);
    double d2 = crossProduct(p3,p4,p2);
    double d3 = crossProduct(p1,p2,p3);
    double d4 = crossProduct(p1,p2,p4);
    if( xy(d1 * d2,0.0) && xy(d3 * d4,0.0) ) return true;
          //如果不判端点相交，则下面这句话不需要
    if( dd(d1,0.0) && onSegment(p3,p4,p1) || dd(d2,0.0) &&
onSegment(p3,p4,p2)
     || dd(d3,0.0) && onSegment(p1,p2,p3) || dd(d4,0.0) &&
onSegment(p1,p2,p4) )
        return true;
    return false;
}
```

5.5.3 判断两直线是否相交

不平行即相交

5.5.4 判断直线和线段是否相交

```
//判断直线和线段是否相交，共线的不算相交
double crossProduct(point a,point b,point c)//向量 ac 在 ab 的方向
bool s2l_inst(point s1,point s2,point l1,point l2)//s 是线段，l 是直线
{    // xyd 包括端点在直线上。xy 是穿过
    return xyd(crossProduct(l1,l2,s1) * crossProduct(l1,l2,s2),0.0);
}
```

5.5.5 判断两向量位置

```
//判断向量 a1a2 与 b1b2 的位置
double rota_angle(point a1,point a2,point b1,point b2) //返回 b1b2 在 a1a2
的方向
{
    point t;
    t.x = b2.x - (b1.x - a1.x);
    t.y = b2.y - (b1.y - a1.y);
    return crossProduct(a1,a2,t);
}
```

5.5.6 求两直线交点

```
//求两直线的交点（先判断相交） ,注意可能得到-0.00000
point l2l_inst_p(point u1,point u2,point v1,point v2)
{
    point ans = u1;
    double t = ((u1.x - v1.x)*(v1.y - v2.y) - (u1.y - v1.y)*(v1.x - v2.x))/
               ((u1.x - u2.x)*(v1.y - v2.y) - (u1.y - u2.y)*(v1.x - v2.x));
    ans.x += (u2.x - u1.x)*t;
    ans.y += (u2.y - u1.y)*t;
    return ans;
}
point l2l_inst_p(line l1,line l2)
```

```
{
    point ans = l1.a;
    double t = ((l1.a.x - l2.a.x)*(l2.a.y - l2.b.y) - (l1.a.y -
l2.a.y)*(l2.a.x - l2.b.x))/
              ((l1.a.x - l1.b.x)*(l2.a.y - l2.b.y) - (l1.a.y -
l1.b.y)*(l2.a.x - l2.b.x));
    ans.x += (l1.b.x - l1.a.x)*t;
    ans.y += (l1.b.y - l1.a.y)*t;
    return ans;
}
```

5.5.7 求两线段交点

```
//求两线段的交点（先判断相交），然后按两直线求交点求得
point s2s_inst_p(point u1,point u2,point v1,point v2)
{
    point ans = u1;
    double t = ((u1.x - v1.x)*(v1.y - v2.y) - (u1.y - v1.y)*(v1.x - v2.x))/
              ((u1.x - u2.x)*(v1.y - v2.y) - (u1.y - u2.y)*(v1.x - v2.x));
    ans.x += (u2.x - u1.x)*t;
    ans.y += (u2.y - u1.y)*t;
    return ans;
}
```

5.5.8 判断点是否在线段上

```
//判断点 c 是否在线段 ab 上。
double crossProduct(point a,point b,point c)
bool onSegment(point a, point b, point c)
{
    double maxx = max(a.x,b.x);
    double maxy = max(a.y,b.y);
    double minx = min(a.x,b.x);
    double miny = min(a.y,b.y);
    if( dd(crossProduct(a,b,c),0.0) && dyd(c.x,minx) && xyd(c.x,maxx) &&
dyd(c.y,miny) && xyd(c.y,maxy) )
        return true;
```

```
    return false;
}
```

5.5.9 点到线的距离，点到线段的最小距离

```
/*************************************************************************
          点到线的距离，点到线段的最小距离
*************************************************************************/
//点到直线的距离
double disp2l(point a,point l1,point l2)
{
    return fabs( crossProduct(a,l1,l2) )/disp2p(l1,l2);
}
//点到线段的最短距离
double crossProduct(point a,point b,point c)
double disp2p(point a,point b)
double disp2seg(point p,point l1,point l2)
{
    point t = p;
    t.x += l1.y - l2.y;  t.y += l2.x - l1.x;
    if( dyd(crossProduct(l1,t,p)*crossProduct(l2,t,p),0.0) ) //包括点和线
段共线
        return xy(disp2p(p,l1),disp2p(p,l2)) ? disp2p(p,l1) : disp2p(p,l2);
    return fabs( crossProduct(p,l1,l2) )/disp2p(l1,l2);
}
```

5.5.10 判断点是否在线段上

```
/*************************************************************************
判断点 c 是否在线段 ab 上。
*************************************************************************/
//判断点 c 是否在线段 ab 上。 包括端点，如果不包括的话 把 xyd -> xy
double crossProduct(point a,point b,point c)
bool onSegment(point a, point b, point c)
{
    if( dd(crossProduct(a,b,c),0.0) && dyd(c.x,min(a.x,b.x)) &&
```

```
        xyd(c.x,max(a.x,b.x)) && dyd(c.y,min(a.y,b.y)) &&
xyd(c.y,max(a.y,b.y)) )
        return true;
    return false;
}
```

5.5.11 求垂点（不是垂足）

```
// 求一个点，使得 ab 垂直于 l1l2
point foot_line(point a,point l1,point l2)    //ac 在 l1l2 的逆时针方向
{
    point c;
    c.x = a.x - l2.y + l1.y;
    c.y = a.y + l2.x - l1.x;
    return c;
}
```

5.5.12 通过两点求直线方程

```
// 通过两点求直线（Ax + By + C = 0）
struct beeline{        double A,B,C;    };
beeline makeline(point a,point b)
{
    beeline line;
    line.A = a.y - b.y;
    line.B = a.x - b.x;
    line.C = -line.A * a.x - line.B*a.y;
    return line;
}
```

5.5.13 向量的旋转

```
//向量的旋转 //底边线段 ab 绕 a 逆时针旋转角度 ang(弧度制)，原 Whirl 函数
```

```
point Rotate(double ang, point a, point b)
{
    b.x -= a.x; b.y -= a.y;
    point c;
    c.x = b.x * cos(ang) - b.y * sin(ang) + a.x;
    c.y = b.x * sin(ang) + b.y * cos(ang) + a.y;
    return c;
}
```

5.5.14 判断点是否在多边形内

```
//判断点是否在多边形内 凸凹均可
double crossProduct(point a,point b,point c)//向量 ac 在 ab 的方向
bool onSegment(point a, point b, point c)
bool s2s_inst(point p1,point p2, point p3, point p4)
// p 中按顺时针或者逆时针方向存多边形的点，n 为多边形的点数
bool point_inPolygon(point pot,point p[],int n)
{
    int count = 0;
    point a = pot,b;
    b.x = 1e20; b.y = pot.y;
    p[n] = p[0];
    for(int i=0; i<n; i++)
    {
        if( onSegment(p[i],p[i+1],pot) ) return true;
        if( !dd(p[i].y,p[i+1].y) )
        {
            int tmp = -1;
            if( onSegment(a,b,p[i]) )
                tmp = i;
            else
                if( onSegment(a,b,p[i+1]) )
                    tmp = i+1;
            if( tmp != -1 && dd(p[tmp].y,max(p[i].y,p[i+1].y)) ||
                tmp == -1 && s2s_inst(p[i],p[i+1],a,b) )
                count++;
        }
    }
    return count % 2 == 1;
}
```

```
//判断点 a 是否在凸多边形内，p 数组至少开[n + 2]个点
bool pin_convexh(point *p,int n,point a)
{
    p[n] = p[0]; p[n+1] = p[1];
    for(int i=0; i<n; i++)
        if( xy(crossProduct(p[i],p[i+1],a)*
            crossProduct(p[i+1],p[i+2],a),0.0) )
            return false;
    return true;
}
```

5.6 线段相交

```
struct point
{
        double x,y;
}fst[105],sec[105];// 线段的两个端点
double dir(point p1,point p2,point p)//叉乘(看点 p 在线段 p1p2 的哪一边)
{
        return (p.x-p1.x)*(p2.y-p1.y)-(p.y-p1.y)*(p2.x-p1.x);
}
bool online(point p1,point p2,point p)//判断点是否是在线段上
{
        if((p.x-p1.x)*(p.x-p2.x)<=0&&(p.y-p1.y)*(p.y-p2.y)<=0)
                return true;
        return false;
}
int intersect(point p1,point p2,point p3,point p4)//判断是否相交
{
        double d1=dir(p3,p4,p1),d2=dir(p3,p4,p2),d3=dir(p1,p2,p3),d4=dir(p1,p2,p4);
        if(d1*d2<0&&d3*d4<0)return 1;
        else if(d1==0&&online(p3,p4,p1))
                return 1;
        else if(d2==0&&online(p3,p4,p2))
                return 1;
        else if(d3==0&&online(p1,p2,p3))
                return 1;
        else if(d4==0&&online(p1,p2,p4))
                return 1;
        return 0;
```

```
}
```

注意：结构体的嵌套会有精度的较大丢失。

向量 AxB>0 则 A 在 B 的顺时针方向，AxB<0 则 A 在 B 的逆时针方向，AxB=0,向量 A 和 B 共线，方向相同或相反。

5.7 质心

求任意多边形的重心

已知一多边形没有边相交，质量分布均匀。顺序给出多边形的顶点坐标，求其重心。

分析：

求多边形重心的题目大致有这么几种：

1，质量集中在顶点上。**n** 个顶点坐标为**(xi,yi)**，质量为 **mi**，则重心

$$X = \Sigma(xi \times mi) / \Sigma mi$$

$$Y = \Sigma(yi \times mi) / \Sigma mi$$

特殊地，若每个点的质量相同，则

$$X = \Sigma xi / n$$

$$Y = \Sigma yi / n$$

2，质量分布均匀。这个题就是这一类型，算法和上面的不同。

特殊地，质量均匀的三角形重心：

$$X = (x0 + x1 + x2) / 3$$

$$Y = (y0 + y1 + y2) / 3$$

3，质量分布不均匀。只能用积分来算，不会**……**

下面讨论多边形的重心

以第一个顶点为基准，分别连接 **p[i]**，**p[i+1]**，**1<i<n** 和任意一点**(**这里取原点**)**将多边形划分为若干个三角形。

若我们求出了每个三角形的重心和质量，可以构造一个新的多边形，顶点为所有三角形的重心，顶点质量为三角形的质量。这个新多边形的质量和重心与原多边形相同，即可使用第一种类型的公式计算出整个多边形的重心。

由于三角形的面积与质量成正比，所以我们这里用面积代替质量来计算。

现在有个问题就是，多边形有可能为凹多边形，三角形有可能在多边形之外。如何处理这种情况呢？

很简单，我们使用叉积来计算三角形面积，当三角形在多边形之外时，得到"负面积"就抵消掉了。

```
struct point//多边形的点
{
	double x,y;
}p[1000005];
struct masspoint//质心
{
	double x,y,weight;
};
void getmass(int n,point &ans)
{
```

```
        ans.x=0,ans.y=0;
        double summass=0;//总质量
        masspoint acr;
        for(int i=0;i<n;i++)
        {
                int j=(i+1)%n;
                acr.x=(p[i].x+p[j].x)/3.0;//三角形的重心 x
                acr.y=(p[i].y+p[j].y)/3.0; //三角形的重心 y
                acr.weight=(p[i].x*p[j].y-p[i].y*p[j].x)/2.0;//该
点的质量
                ans.x+=acr.x*acr.weight;ans.y+=acr.y*acr.weight;
                summass+=acr.weight;
        }
        ans.x/=summass;ans.y/=summass;
        if(fabs(ans.x)<0.005) ans.x=0;//防止-0.00 的情况出现
    if(fabs(ans.y)<0.005) ans.y=0;
        printf("%.2f %.2f\n",ans.x,ans.y);
}
```

5.8 最近点对

```
const int size=100005,MAX=1<<30,MIN=-(1<<30);
struct point
{
        double x,y;
}X[size],Y[size];
bool cmpx(point a,point b)
{
        return a.x<b.x;
}
bool cmpy(point a,point b)
{
        return a.y<b.y;
}
double dis(point a,point b)
{
        return sqrt((a.x-b.x)*(a.x-b.x)+(a.y-b.y)*(a.y-b.y));
}
double min_dis(int l,int r)//以 y 为主划分，x 为辅划分
{
        double ans=1e8;
        if(r-l<=2)//小于等于 3 个点
        {
```

```
                for(int i=l;i<r;i++)
                        for(int j=i+1;j<=r;j++)
                                ans=min(ans,dis(Y[i],Y[j]));
        }
        else
        {
                int len=0,mid=(l+r)/2;
                ans=min(min_dis(l,mid),min_dis(mid+1,r));//分治
                for(int i=l;i<=r;i++)//删除不可能的点
                {
                        if(fabs(Y[i].y-Y[mid].y)<ans)
                                X[len++]=Y[i];
                        else if(i>mid)break;//后面的都不满足
                }
                sort(X,X+len,cmpx);对这些点关于 X 再排序
                for(int i=0;i<len-1;i++)
                   for(int j=i+1;j<len;j++)//少于等于 7 个点写成 j<=i+8&&j<
len 也可
                   {
                           if(fabs(X[i].x-X[j].x)<ans)
                                   ans=min(ans,dis(X[i],X[j]));
                           else break; //后面的都不满足
                   }
        }
        return ans;
}
int main()
{
        int n;
        while(~scanf("%d",&n))
        {
                if(n==0)break;
                for(int i=0;i<n;i++)
                        scanf("%lf%lf",&Y[i].x,&Y[i].y);
                sort(Y,Y+n,cmpy);//先按 Y 排序
                printf("%.2f\n",min_dis(0,n-1)/2.0);
        }
        return 0;
}
```

5.9 凸包

```
struct point                            //求凸包
```

```
{
        double x,y;
}p[105],stack[105];
int top;
double multiply(point p,point p1,point p2)
{
        return (p1.x-p.x)*(p2.y-p.y)-(p2.x-p.x)*(p1.y-p.y);
}
double dis(point a,point b)
{
        return sqrt((a.x-b.x)*(a.x-b.x)+(a.y-b.y)*(a.y-b.y));
}
bool cmp(point a,point b)
{
        double m=multiply(p[0],a,b);//叉乘求方向
        if(m>0)return true;
        else if(m==0&&dis(a,p[0])<dis(b,p[0]))共线则取短的
                return true;
        return false;
}
void graham(int n)
{
        int bin;
        double x=1e8,y=1e8;
        for(int i=0;i<n;i++)//寻找最下方的点
        {
                if(p[i].y<y||(p[i].y==y&&p[i].x<x))
                        bin=i,x=p[i].x,y=p[i].y;
        }
        swap(p[0],p[bin]);//p[0]保存最下方的点
        sort(p+1,p+n,cmp);
        top=0;
        for(int i=0;i<3;i++)前三个进栈
                stack[top++]=p[i];
        for(int i=3;i<n;i++)
        {
                while(top>=2&&multiply(stack[top-2],stack[top-1],p[i])<=
0)
//不满足凸包则退栈
                        top--;
                stack[top++]=p[i];//加入 p[i];
        }
}
```

Stack 保存的就是凸包的所有点，逆时针的。

5.10 三角形

5.10.1 计算三角形面积，边长

```
/*************************************************************************
            计算三角形面积||边长，已知三高 || 三点 || 三边 || 三中线
*************************************************************************/

//计算三角形面积（已知三角形的三条高）
const double eps = 1e-8; // 精度要高
double area_triangle_h(double x,double y,double z)
{
    if( dd(x,0.0) || dd(y,0.0) || dd(z,0.0) ) return -1; // 构不成三角形
    double p =  (1/x +  1/y + 1/z)*(1/y + 1/z - 1/x)*(1/x + 1/z - 1/y)*(1/x
+ 1/y - 1/z);
    if( xyd(p,0.0) ) return -1; // 构不成三角形
    return 1/sqrt(p);
}
//计算三角形面积(三点)
double area_triangle(point a,point b,point c)
{
    return fabs( crossProduct(a,b,c) )/2.0;
}
//计算三角形面积（三边海伦公式）
double area_triangle(double a,double b,double c)
{
    double p = (a+b+c)/2.0;
    return sqrt(p*(p-a)*(p-b)*(p-c));
}
//计算三角形面积（三点，坐标）
double area_triangle(double x0,double y0,double x1,double y1,double
x2,double y2)
{
    return fabs( crossProduct(x0,y0,x1,y1,x2,y2) )/2.0;
}
//计算三角形面积（已知三条中线长）
double area_midline(double x,double y,double z)
```

```
{
    return 4.0/3*area_triangle(x,y,z);
}
//公式 已知中线 x y z 求边长 a b c
void triangle_edge(double x,double y,double z,double &a,double &b,double
&c)
{
    a = 2.0/3.0*sqrt(2*x*x + 2*z*z - y*y);
    b = 2.0/3.0*sqrt(2*y*y + 2*z*z - x*x);
    c = 2.0/3.0*sqrt(2*x*x + 2*y*y - z*z);
}
```

5.10.2 计算三角形的外接圆半径，内切圆半径

```
double circumcir_r(double a,double b,double c)//已知三边求外接圆半径
{
    return a*b*c/(4*area_triangle(a,b,c));
}
double  circumcir_r(point ap,point bp,point cp)//已知三点求外接圆半径
{
    double a,b,c;
    a = disp2p(ap,bp); b = disp2p(bp,cp); c = disp2p(cp,ap);
    return a*b*c/sqrt((a+b+c)*(a+b-c)*(a+c-b)*(b+c-a));
}

double incir_r(double a,double b,double c)//已知三边求内切圆半径
{
    return 2*area_triangle(a,b,c)/(a+b+c);
}
double incir_r(point ap,point bp,point cp)//已知三点求内切圆半径
{
    double a,b,c;
    a = disp2p(ap,bp); b = disp2p(bp,cp); c = disp2p(cp,ap);
    return area_3p(ap,bp,cp)*2.0/(a+b+c);
}
```

5.10.3 各种心（外心，内心，垂心，重心，费马点）

```
//计算三角形重心
//到三角形三点距离的平方和最小的点
//三角形内到三边距离之积最大的点
point l2l_inst_p(point u1,point u2,point v1,point v2)
point barycenter(point a,point b,point c)
{
    point u1,u2,v1,v2;
    u1.x = (a.x + b.x)/2;
    u1.y = (a.y + b.y)/2;
    v1.x = (a.x + c.x)/2;
    v1.y = (a.y + c.y)/2;
    return l2l_inst_p(u1,c,v1,b);
}
//三角形的内心
point incenter(point a,point b,point c)
{
    line u,v;
    double m,n;
    u.a = a;
    m = atan2(b.y - a.y, b.x - a.x);
    n = atan2(c.y - a.y, c.x - a.x);
    u.b.x = u.a.x + cos((m+n)/2);
    u.b.y = u.a.y + sin((m+n)/2);
    v.a = b;
    m = atan2(a.y - b.y, a.x - b.x);
    n = atan2(c.y - b.y, c.x - b.x);
    v.b.x = v.a.x + cos((m+n)/2);
    v.b.y = v.a.y + sin((m+n)/2);
    return l2l_inst_p(u,v);
}
//计算三角形外心（外接圆圆心）
point l2l_inst_p(line u,line v)
point circumcenter(point a,point b,point c)
{
    point ua,ub,va,vb;
    ua.x = ( a.x + b.x )/2;
    ua.y = ( a.y + b.y )/2;
    ub.x = ua.x - a.y + b.y;//根据 垂直判断，两线段点积为 0
    ub.y = ua.y + a.x - b.x;
```

```
    va.x = ( a.x + c.x )/2;
    va.y = ( a.y + c.y )/2;
    vb.x = va.x - a.y + c.y;
    vb.y = va.y + a.x - c.x;
    return l2l_inst_p(ua,ub,va,vb);
}
//这种不需要相交的那个函数 （外接圆圆心）
point circumcenter(point a,point b,point c)
{
     point ret;
     double a1 = b.x - a.x, b1 = b.y - a.y, c1 = (a1*a1 + b1*b1)/2;
      double a2 = c.x - a.x, b2 = c.y - a.y, c2 = (a2*a2 + b2*b2)/2;
      double d = a1 * b2 - a2 * b1;
      ret.x = a.x + (c1*b2 - c2*b1)/d;
      ret.y = a.y + (a1*c2 - a2*c1)/d;
 return ret;
}
//三角形的垂心（垂线的交点）
point perpencenter(point a,point b,point c)
{
    point ua,ub,va,vb;
    ua = c;
    ub.x = ua.x - a.y + b.y;
    ub.y = ua.y + a.x - b.x;
    va = b;
    vb.x = va.x - a.y + c.y;
    vb.y = va.y + a.x - c.x;
    return l2l_inst_p(ua,ub,va,vb);
}
//费马点
//到三角形三顶点距离之和最小的点
point fermentpoint(point a,point b,point c){
    point u,v;
    double
step=fabs(a.x)+fabs(a.y)+fabs(b.x)+fabs(b.y)+fabs(c.x)+fabs(c.y);
    int i,j,k;
    u.x=(a.x+b.x+c.x)/3;
    u.y=(a.y+b.y+c.y)/3;
    while (step>1e-10)
        for (k=0;k<10;step/=2,k++)
            for (i=-1;i<=1;i++)
                for (j=-1;j<=1;j++){
                    v.x=u.x+step*i;
                    v.y=u.y+step*j;
```

```
                if
(distance(u,a)+distance(u,b)+distance(u,c)>distance(v,a)+distance(v,b)+
distance(v,c))
                    u=v;
            }
    return u;
}
```

5.11 圆

5.11.1 圆的位置关系

```
/*************************************************************************
圆的位置关系：判断 相交 相切 内含
                        求 交点 切点
          圆和矩形是否相交
*************************************************************************/
//判断两圆是否相交 (不包括相切)
bool c2c_inst(point a,double r1,point b,double r2)
{
    if( xy(disp2p(a,b),r1+r2) && dy(disp2p(a,b),fabs(r1 - r2)) )
        return true;
    return false;
}
//求 直线与圆的交点（保证有交点）
// 也可计算 线段与圆的交点，然后求得交点判断是否在线段上
void l2c_inst_p(point c,double r,point l1,point l2,point &p1,point &p2)
{
    point p = c;
    double t;
    p.x += l1.y - l2.y;
    p.y += l2.x - l1.x;
    p = l2l_inst_p(p,c,l1,l2);
    t = sqrt(r*r - disp2p(p,c)*disp2p(p,c))/disp2p(l1,l2);
    p1.x = p.x + (l2.x - l1.x)*t;
    p1.y = p.y + (l2.y - l1.y)*t;
    p2.x = p.x - (l2.x - l1.x)*t;
    p2.y = p.y - (l2.y - l1.y)*t;
```

```
}
//求两圆交点（先判断相交）
//交得弧是弧 p1->p2
void c2c_inst_p(point c1,double r1,point c2,double r2,point &p1,point &p2)
{
    point u,v;
    double t;
    t = (1 + (r1*r1 - r2*r2)/disp2p(c1,c2)/disp2p(c1,c2))/2;
    u.x = c1.x + (c2.x - c1.x)*t;
    u.y = c1.y + (c2.y - c1.y)*t;
    v.x = u.x + c1.y - c2.y;
    v.y = u.y - c1.x + c2.x;
    l2c_inst_p(c1,r1,u,v,p1,p2);
}

//判断两圆是否相切 （外切和内切）
bool c2c_tangent(point a,double r1,point b,double r2)
{
    if( dd(disp2p(a,b),r1+r2) || dd(disp2p(a,b),fabs(r1-r2)) )
        return true;
    return false;
}
//求两圆切点（先判断相切，内外切交点不一样哎）
point c2c_tangent_p(point a,double r1,point b,double r2)
{
    point t;
    if( dd(disp2p(a,b),r1 + r2)  ) // 外切交点
    {
        t.x = (r1*b.x + r2*a.x)/(r1 + r2);
        t.y = (r1*b.y + r2*a.y)/(r1 + r2);
        return t;
    }
    t.x = (r1*b.x - r2*a.x)/(r1 - r2);
    t.y = (r1*b.y - r2*a.y)/(r1 - r2);
    return t;
}
//判断两圆是否内含（a 内含于 b，不包括相切）
bool c2c_ainb(point a,double r1,point b,double r2)
{
    return xy(disp2p(a,b),r2 - r1);    //a 在 b 中，如果是包括内切，用 xyd
}
```

5.11.2 两圆相交面积

```
double gongxing_area(double r1,double r2,double l) // 两圆半径以及连心线长度
{
    double cosaa = (r1*r1 + l*l - r2*r2)/(2*r1*l);      // cos 值
    double fcosaa = acos(cosaa)*2;                      //夹角
    double Shu = fcosaa*r1*r1/2;                        // 扇形面积
    double Ssan = r1*r1*sin(fcosaa)/2;                  // 三角形面积
    return (Shu - Ssan);                                //扇形面积减去三角形面积
}
double c2c_inst_area(point a,double r1,point b,double r2)
{
    if( c2c_ainb(a,r1,b,r2) )    return pi*r1*r1;    // a 内含于 b
    if( c2c_ainb(b,r2,a,r1) )    return pi*r2*r2;    // b 内含于 a
    if( !c2c_inst(a,r1,b,r2) )    return 0;
    double l = disp2p(a,b);                 // 如果单纯求相交面积，需判断相交，从这儿到下面这段
    return gongxing_area(r1,r2,l) + gongxing_area(r2,r1,l);
}
```

5.11.3 判断圆和矩形是否相交

```
//判断圆和矩形是否相交 ,包括只交一个点
bool c2r_inst(point a,double r,point p[])
{
    bool flag = false;
    for(int i=0; i<4; i++)//如果严格相交，不要等号
        if( dyd(disp2p(a,p[i]),r) )
            flag = true;
    if( !flag ) return false;
    flag = false;
    for(int i=0; i<4; i++)
        if( xyd(disp2seg(a,p[i],p[(i+1)%4]),r) )
            flag = true;
    if( !flag ) return false;
```

```
    return true;
}
```

5.11.4 最小覆盖圆

```
/*********************************************************************
                        n 个点的最小覆盖圆 O(n)算法
*********************************************************************/
point circumcenter(point a,point b,point c)
void min_cover_circle(point p[],int n,point &c,double &r)
{
    random_shuffle(p,p+n);// #include <algorithm>
    c = p[0]; r = 0;
    for(int i=1; i<n; i++)
        if( dy(disp2p(p[i],c),r) )
        {
            c = p[i]; r = 0;
            for(int k=0; k<i; k++)
                if( dy(disp2p(p[k],c),r) )
                {
                    c.x = (p[i].x + p[k].x)/2;
                    c.y = (p[i].y + p[k].y)/2;
                    r = disp2p(p[k],c);
                    for(int j=0; j<k; j++)
                        if( dy(disp2p(p[j],c),r) )
                        {                          // 求外接圆圆心，三点必不
共线
                            c = circumcenter(p[i],p[k],p[j]);
                            r = disp2p(p[i],c);
                        }
                }
        }
}
```

5.11.5 扇形重心距圆心距离

```
// 扇形重心距圆心距离
```

```
//Xc = 2*R*sinA/3/A
//A 为圆心角的一半
double dis_z2c(double r, double angle)
{
    return 2 * r * sin(angle) / 3 / angle;
}
```

5.12 多边形

5.12.1 判断两个矩形是否相交

```
//判断两个矩形是否相交（即有公共面积），用左下角和右上角点表示矩形
bool r2r_inst(point a1,point a2,point b1,point b2)
{
    if( b1.x >= a2.x || b2.x <= a1.x || b1.y >= a2.y || b2.y <= a1.y )
        return false;
    return true;
}
```

5.12.2 多边形面积

```
// 计算多边形的面积，可以根据面积判断是哪种方向
//按照我写的这个，顺时针的话，没有加绝对值的面积为负
double area_polygon(point p[],int n)
{
    if( n < 3 ) return 0.0;
    double s = 0.0;
    for(int i=0; i<n; i++)
        s += p[(i+1)%n].y * p[i].x - p[(i+1)%n].x * p[i].y;
    return fabs(s)/2.0;
}
//改变时针方向
void change_wise(point p[],int n)
{
    for(int i=0; i<n/2; i++)
```

```
        swap(p[i],p[n-i-1]);
}
```

5.12.3 多边形的重心

```
//计算多边形的重心（凸凹均可）
double crossProduct(point a,point b,point c)//向量 ac 在 ab 的方向
point bary_center(point p[],int n)
{
    point ans,t;
    double area = 0.0,t2;
    ans.x = 0.0; ans.y = 0.0;
    for(int i=1; i<n-1; i++)
    {
        t2 = crossProduct(p[i],p[0],p[i+1])/2.0;
        ans.x += (p[0].x + p[i].x + p[i+1].x)*t2;
        ans.y += (p[0].y + p[i].y + p[i+1].y)*t2;
        area += t2;
    }
    ans.x /= (3*area);
    ans.y /= (3*area);
    return ans;
}
```

5.13 凸包相关

5.13.1 二维凸包

```
//判断一个多边形是否是凸包，判断线旋转方向即可
bool is_convexhull(point p[],int n)
{
    for(int i=0; i<n; i++)
        if( xy( crossProduct(p[i],p[(i+1)%n],p[(i+2)%n]) *
            crossProduct(p[(i+1)%n],p[(i+2)%n],p[(i+3)%n]),0.0) )
```

```
            return false;
    return true;
}

//凸包无序排序 s 为内点
point s;
bool cmp(point a,point b)
{
    double l1 = crossProduct(s,c[0],a);
    double l2 = crossProduct(s,c[0],b);
    if( dyd(l1,0.0) && dyd(l2,0.0) )
        return xy(crossProduct(s,a,b),0.0);
    if( dyd(l1,0.0) && xyd(l2,0.0) )
        return 0;
    if( xyd(l1,0.0) && xyd(l2,0.0) )
        return xy(crossProduct(s,a,b),0.0);
    return 1;
}
void sort_chull()
{
    s.x = (c[0].x + 2 * c[1].x + c[2].x)/4;  // S 求法：线的中点，俩中点再求
中点
    s.y = (c[0].y + 2 * c[1].y + c[2].y)/4;
    int tmp = 0;
    for(int i=1; i<n; i++)
        if( xy(c[i].x,c[tmp].x) || dd(c[i].x,c[tmp].x) &&
xy(c[i].y,c[tmp].y) )
            tmp = i;
    swap(c[tmp],c[0]);
    sort(c+1,c+n,cmp);
}
/*********************************************************************
                                求凸包
*********************************************************************/
double crossProduct(point a,point b,point c)
double disp2p(point a,point b)
bool cmp(point a,point b)  // 排序
{
    double len = crossProduct(c[0],a,b);
    if( dd(len,0.0) )
        return xy(disp2p(c[0],a),disp2p(c[0],b));
    return xy(len,0.0);
}
int stk[MAX];
```

```
int top;
//形成凸包条件：输入点个数大于等于三且至少有三个点不共线
//纯净凸包（即 stk 中没有三点共线）
void Graham(int n)
{
    int tmp = 0;
    for(int i=1; i<n; i++)
        if( xy(c[i].x,c[tmp].x) || dd(c[i].x,c[tmp].x) &&
xy(c[i].y,c[tmp].y) )
            tmp = i;
    swap(c[0],c[tmp]);
    sort(c+1,c+n,cmp);
    stk[0] = 0; stk[1] = 1;
    top = 1;
    for(int i=2; i<n; i++)
    {
        while( xyd( crossProduct(c[stk[top]],c[stk[top-1]],c[i]), 0.0 ) &&
top >= 1 )
            top--;
        stk[++top] = i;
    }
}
void Graham(int n)
{
    int tmp = 0;
    for(int i=1; i<n; i++)
        if( xy(c[i].x,c[tmp].x) || dd(c[i].x,c[tmp].x) &&
xy(c[i].y,c[tmp].y) )
            tmp = i;
    swap(c[0],c[tmp]);
    sort(c+1,c+n,cmp);
    stk[0] = c[0]; stk[1] = c[1];
    top = 1;
    for(int i=2; i<n; i++)
    {
        while( xyd( crossProduct(stk[top],stk[top-1],c[i]), 0.0 ) && top >=
1 )
            top--;
        stk[++top] = c[i];
    }
}
```

5.13.2 三维凸包

```
//三维凸包模板,可以求三维凸包表面积,体积,表面多边形数和表面三角形数
#define eps 1e-7
#define MAXV 310

//三维点
struct pt{
    double x, y, z;
    pt(){}
    pt(double _x, double _y, double _z): x(_x), y(_y), z(_z){}
    pt operator - (const pt p1){return pt(x - p1.x, y - p1.y, z - p1.z);}
    pt operator * (pt p){return pt(y*p.z-z*p.y, z*p.x-x*p.z, x*p.y-y*p.x);}
//叉乘
    double operator ^ (pt p){return x*p.x+y*p.y+z*p.z;}
//点乘
};

struct _3DCH{
    struct fac{
        int a, b, c;    //表示凸包一个面上三个点的编号
        bool ok;        //表示该面是否属于最终凸包中的面
    };

    int n;    //初始点数
    pt P[MAXV];    //初始点

    int cnt;    //凸包表面的三角形数
    fac F[MAXV*8]; //凸包表面的三角形

    int to[MAXV][MAXV];

    double vlen(pt a){return sqrt(a.x*a.x+a.y*a.y+a.z*a.z);}    //向量长度
    double area(pt a, pt b, pt c){return vlen((b-a)*(c-a));}    //三角形面
积*2
    double volume(pt a, pt b, pt c, pt d){return (b-a)*(c-a)^(d-a);}    //
四面体有向体积*6

    //正：点在面同向
    double ptof(pt &p, fac &f){
        pt m = P[f.b]-P[f.a], n = P[f.c]-P[f.a], t = p-P[f.a];
```

```
        return (m * n) ^ t;
    }

    void deal(int p, int a, int b){
        int f = to[a][b];
        fac add;
        if (F[f].ok){
            if (ptof(P[p], F[f]) > eps)
                dfs(p, f);
            else{
                add.a = b, add.b = a, add.c = p, add.ok = 1;
                to[p][b] = to[a][p] = to[b][a] = cnt;
                F[cnt++] = add;
            }
        }
    }

    void dfs(int p, int cur){
        F[cur].ok = 0;
        deal(p, F[cur].b, F[cur].a);
        deal(p, F[cur].c, F[cur].b);
        deal(p, F[cur].a, F[cur].c);
    }

    bool same(int s, int t){
        pt &a = P[F[s].a], &b = P[F[s].b], &c = P[F[s].c];
        return fabs(volume(a, b, c, P[F[t].a])) < eps && fabs(volume(a, b,
c, P[F[t].b])) < eps && fabs(volume(a, b, c, P[F[t].c])) < eps;
    }

    //构建三维凸包
    void construct(){
        cnt = 0;
        if (n < 4)
            return;

        /*********此段是为了保证前四个点不公面，若已保证，可去掉********/
        bool sb = 1;
        //使前两点不公点
        for (int i = 1; i < n; i++){
            if (vlen(P[0] - P[i]) > eps){
                swap(P[1], P[i]);
                sb = 0;
                break;
```

```
        }
    }
    if (sb)return;

    sb = 1;
    //使前三点不公线
    for (int i = 2; i < n; i++){
        if (vlen((P[0] - P[1]) * (P[1] - P[i])) > eps){
            swap(P[2], P[i]);
            sb = 0;
            break;
        }
    }
    if (sb)return;

    sb = 1;
    //使前四点不共面
    for (int i = 3; i < n; i++){
        if (fabs((P[0] - P[1]) * (P[1] - P[2]) ^ (P[0] - P[i])) > eps){
            swap(P[3], P[i]);
            sb = 0;
            break;
        }
    }
    if (sb)return;
    /*********此段是为了保证前四个点不公面********/


    fac add;
    for (int i = 0; i < 4; i++){
        add.a = (i+1)%4, add.b = (i+2)%4, add.c = (i+3)%4, add.ok = 1;
        if (ptof(P[i], add) > 0)
            swap(add.b, add.c);
        to[add.a][add.b] = to[add.b][add.c] = to[add.c][add.a] = cnt;
        F[cnt++] = add;
    }

    for (int i = 4; i < n; i++){
        for (int j = 0; j < cnt; j++){
            if (F[j].ok && ptof(P[i], F[j]) > eps){
                dfs(i, j);
                break;
            }
        }
```

```
        }
        int tmp = cnt;
        cnt = 0;
        for (int i = 0; i < tmp; i++){
            if (F[i].ok){
                F[cnt++] = F[i];
            }
        }
    }

    //表面积
    double area(){
        double ret = 0.0;
        for (int i = 0; i < cnt; i++){
            ret += area(P[F[i].a], P[F[i].b], P[F[i].c]);
        }
        return ret / 2.0;
    }

    //体积
    double volume(){
        pt O(0, 0, 0);
        double ret = 0.0;
        for (int i = 0; i < cnt; i++){
            ret += volume(O, P[F[i].a], P[F[i].b], P[F[i].c]);
        }
        return fabs(ret / 6.0);
    }

    //表面三角形数
    int facetCnt_tri(){
        return cnt;
    }

    //表面多边形数
    int facetCnt(){
        int ans = 0;
        for (int i = 0; i < cnt; i++){
            bool nb = 1;
            for (int j = 0; j < i; j++){
                if (same(i, j)){
                    nb = 0;
                    break;
                }
```

```
            }
            ans += nb;
        }
        return ans;
    }
};

_3DCH hull;    //内有大数组，不易放在函数内

int main()
{
    while (~scanf("%d", &hull.n)){
        for (int i = 0; i < hull.n; i++)
            scanf("%lf%lf%lf", &hull.P[i].x, &hull.P[i].y, &hull.P[i].z);
        hull.construct();
        printf("%d\n", hull.facetCnt());
    }
    return 0;
}
```

5.14 旋转卡壳

```
//旋转卡壳 先求凸包(纯净)，得出 stk 数组，n 为 top+1

// 返回最远点对距离
double crossProduct(Point a,Point b,Point c)//向量 ac 在 ab 的方向
double RC_maxdisp2p_inhull(int stk[],int n)
{
    stk[n] = stk[0];
    int q = 1;
    double ans = 0.0;
    for(int i=0; i<n; i++)
    {
        while( xy(fabs(crossProduct(c[stk[i]],c[stk[i+1]],c[stk[q]])),

fabs(crossProduct(c[stk[i]],c[stk[i+1]],c[stk[(q+1)%n]])) )
            q = (q+1)%n;
        ans = max(ans,disp2p(c[stk[i]],c[stk[q]]));
    }
    return ans;
```

```
}

//旋转卡壳求两个凸包之间最短距离
double crossProduct(point a,point b,point c)
double disp2p(point a,point b)
bool cmp(point a,point b)  // 凸包排序
void Graham(point c[],point stk[],int &top,int n) // 凸包
double disp2seg(point p,point l1,point l2)// 点到线段的最短距离
double rota_angle(point a1,point a2,point b1,point b2) //返回 b1b2 在 a1a2
的方向
double RC_mindish2h(point p[],point q[],int np,int nq) // 返回最短距离
{
    int sp = 0,sq = 0;
    for(int i=1; i<np; i++)
        if( xy(p[i].y,p[sp].y) || dd(p[sp].y,p[i].y) &&
xy(p[i].x,p[sp].x) )
            sp = i;            //求得第一个凸包最左下角的点

    for(int i=1; i<nq; i++)
        if( dy(q[i].y,q[sq].y) || dd(q[sq].y,q[i].y) &&
dy(q[i].x,q[sq].x) )
            sq = i;            //求得第二个凸包最右上角的点

    int tp = sp,tq = sq;
    double ans = disp2p(p[sp],q[sq]);
    do
    {
        double len = rota_angle(p[sp],p[(sp+1)%np],q[(sq+1)%nq],q[sq]);
        if( dd(len,0.0) )                  // 卡壳正好和俩凸包的边重合
        {
            ans = min(ans,disp2seg(p[sp],q[sq],q[(sq+1)%nq]));
            ans = min(ans,disp2seg(p[(sp+1)%np],q[sq],q[(sq+1)%nq]));
            ans = min(ans,disp2seg(q[sq],p[sp],p[(sp+1)%np]));
            ans = min(ans,disp2seg(q[(sq+1)%nq],p[sp],p[(sp+1)%np]));
            sp++; sp %= np; sq++; sq %= nq;
        }
        else
            if( xy(len,0.0) )              // 卡壳和第一个凸包的边重合
            {
                ans = min(ans,disp2seg(q[sq],p[sp],p[(sp+1)%np]));
                sp++; sp %= np;
            }
            else
            {
```

```
                ans = min(ans,disp2seg(p[sp],q[sq],q[(sq+1)%nq]));// 卡壳和第二个凸包的边重合
                sq++; sq %= nq;
            }
    }while( !(tp == sp && tq == sq) );
    return ans;
}

//求凸包最小面积外接矩形
double crossProduct(point a,point b,point c)//向量 ac 在 ab 的方向
double disp2p(point a,point b)
bool cmp(point a,point b)  // 凸包排序
double disp2seg(point a,point l1,point l2)
point foot_line(point a,point l1,point l2)    // 求一个点，使得 ab 垂直于 l1l2
double rota_angle(point a1,point a2,point b1,point b2) //判断向量 a1a2 与 b1b2 的位置
double Graham(int n)
double RC_minareaRectangle(point p[],int n) //返回面积
{
    int r[4];         // 0 == ymin, 1 == xmin, 2 == ymax ,3 == xmax;
    memset(r,0,sizeof(r));
    for(int i=0; i<n; i++)
    {
        if( xy(p[i].y,p[r[0]].y) )    r[0] = i;
        if( xy(p[i].x,p[r[1]].x) )    r[1] = i;
        if( dy(p[i].y,p[r[2]].y) )    r[2] = i;
        if( dy(p[i].x,p[r[3]].x) )    r[3] = i;
    }
    int tp = r[0];
    double area = inf;
    do
    {
        point t = foot_line(p[r[0]],p[r[0]],p[(r[0]+1)%n]);
        while( dy(rota_angle(t,p[r[0]],p[r[1]],p[(r[1]+1)%n]),0.0) )
            r[1]++, r[1] %= n;

        while( dy(rota_angle(p[r[0]],t,p[r[3]],p[(r[3]+1)%n]),0.0) )
            r[3]++, r[3] %= n;

        while( dy(disp2seg(p[(r[2]+1)%n],p[r[0]],p[(r[0]+1)%n]),
                disp2seg(p[r[2]],p[r[0]],p[(r[0]+1)%n])) )
            r[2]++, r[2] %= n;

        double a = disp2seg(p[r[2]],p[r[0]],p[(r[0]+1)%n]);
```

```
        t = foot_line(p[r[3]],p[r[0]],p[(r[0]+1)%n]);
        double b = disp2seg(p[r[1]],p[r[3]],t);
        area = min( area, a*b );
        r[0]++; r[0] %= n;
    }while( r[0] != tp );
    return area;
}
//卡壳求最大三角形面积，凸包预处理
double RC(point *s,int n)    // 点集是凸包
{
    int p,q,r;
    p = 0; q = 1; r = 2;
    double area = area_triangle(s[p],s[q],s[r]);
    for(p=0; p<n; p++)
    {
        q = (p+1)%n; r = (p+2)%n;
        area = max(area,area_triangle(s[p],s[q],s[r]));
        while( xy(fabs(crossProduct(s[p],s[q],s[r])),
                fabs(crossProduct(s[p],s[q],s[(r+1)%n]))) && r != p )
        {
            area = max(area,area_triangle(s[p],s[q],s[(r+1)%n]));
            r = (r+1)%n;
        }
        if( r == p ) continue;
        int rr = (r+1)%n;
        while( q != rr && r != p )
        {
            area = area = max(area,area_triangle(s[p],s[q],s[r]));
            while( xy(fabs(crossProduct(s[p],s[q],s[r])),
                fabs(crossProduct(s[p],s[q],s[(r+1)%n]))) && r != p )
                    r = (r+1)%n;
            q = (q+1)%n;
        }
    }
    return area;
}
```

5.15 半平面交

```
// 线段 ab 向左方向推进 h 后得到 cd
```

```
void changepoint(point a,point b,point &c,point &d,double h)
{
    double len = disp2p(a,b);
    double dx = h / len * ( a.y - b.y );
    double dy = h / len * (-a.x + b.x );
    c.x = a.x + dx; c.y = a.y + dy;
    d.x = b.x + dx; d.y = b.y + dy;
}

//半平面交 N^2 算法，注意初始化有界平面（初始化成矩形，p[n]=p[0]）
//每增加一条直线 ab 切割  调用这个函数。最终切点存在 s 中，长度为 len。
void cut_hp(point a,point b,point *s,int &len)
{
    int tc = 0;
    point tp[MAX];
    for(int i=0; i<=len; i++)
        tp[i] = s[i];
    for(int i=0; i<len; i++)
    {
        if( xyd(crossProduct(a,b,tp[i]),0.0) )//右侧区域的话是 dyd
            s[tc++] = tp[i];
        if( xy(crossProduct(a,b,tp[i])*
               crossProduct(a,b,tp[i+1]),0.0) )
            s[tc++] = l2l_inst_p(a,b,tp[i],tp[i+1]);
    }
    s[tc] = s[0];
    len = tc;
}

// 半平面交 N*LOGN 注意初始化有界平面
// 半平面返回点集 s，点集点的个数 len
//ln 是 n 个半平面，用 line 的两个端点表示，一律考虑线段左边是半平面
//如果 ln 中有线段右边是半平面的，可以交换线段两点坐标
point l2l_inst_p(line l1,line l2)
bool parallel(line u,line v)
bool equal_ang(line a,line b)    // 第一次 unique 的比较函数
{
    return dd(a.ang,b.ang);
}
bool cmphp(line a,line b)    // 排序的比较函数
{
    if( dd(a.ang,b.ang) ) return xy(crossProduct(b.a,b.b,a.a),0.0);
    return xy(a.ang,b.ang);
}
```

```
bool equal_p(point a,point b)//第二次 unique 的比较函数
{
    return dd(a.x,b.x) && dd(a.y,b.y);
}
void makeline_hp(double x1,double y1,double x2,double y2,line &l)
{
    l.a.x = x1; l.a.y = y1; l.b.x = x2; l.b.y = y2;
    l.ang = atan2(y2 - y1,x2 - x1);
}
void makeline_hp(point a,point b,line &l) // 线段(向量 ab)左侧侧区域有效
{
    l.a = a; l.b = b;
    l.ang = atan2(b.y - a.y,b.x - a.x);    // 如果是右侧区域,改成 a.y - b.y,a.x
- b.x
}
void inst_hp_nlogn(line *ln,int n,point *s,int &len)
{
    len = 0;
    sort(ln,ln+n,cmphp);
    n = unique(ln,ln+n,equal_ang) - ln;
    int bot = 0,top = 1;
    deq[0] = ln[0]; deq[1] = ln[1];
    for(int i=2; i<n; i++)
    {
        if( parallel(deq[top],deq[top-1]) ||
parallel(deq[bot],deq[bot+1]) )
            return ;
        while( bot < top && dy(crossProduct(ln[i].a,ln[i].b,
            l2l_inst_p(deq[top],deq[top-1])),0.0) )
            top--;
        while( bot < top && dy(crossProduct(ln[i].a,ln[i].b,
            l2l_inst_p(deq[bot],deq[bot+1])),0.0) )
            bot++;
        deq[++top] = ln[i];
    }
    while( bot < top && dy(crossProduct(deq[bot].a,deq[bot].b,
        l2l_inst_p(deq[top],deq[top-1])),0.0) )    top--;
    while( bot < top && dy(crossProduct(deq[top].a,deq[top].b,
        l2l_inst_p(deq[bot],deq[bot+1])),0.0) )    bot++;
    if( top <= bot + 1 ) return ;

    for(int i=bot; i<top; i++)
        s[len++] = l2l_inst_p(deq[i],deq[i+1]);
    if( bot < top + 1 ) s[len++] = l2l_inst_p(deq[bot],deq[top]);
```

```
    len = unique(s,s+len,equal_p) - s;
}
```

5.16 整点相关

```
//计算多边形边的整点（包括顶点）
int gcd(int n,int m)
int intp_insegment(point a, point b)
int intp_edge(point p[],int n)
{
    int ans = n;
    for(int i=0; i<n; i++)
        ans += intp_insegment(p[i], p[(i+1)%n]);
    return ans;
}
// 求多边形内的整点个数，多边形顶点都是整点&& dx dy 不能为 0
double area_polygon(point p[],int n)
int intp_inpolygon(point p[],int n)
{
    double area = area_polygon(p,n);
    int pinedge = intp_edge(p,n);
    return (int)(area) - pinedge/2 + 1;
}
//计算在线段上的整点个数 （还是这个比较好）
int intp_insegment(point a, point b)
{
    int aa = abs(b.y - a.y), bb = abs(b.x - a.x);
    if(aa == 0 && bb == 0)    return 0;
    if(aa == 0)    return bb - 1;
    if(bb == 0)    return aa - 1;
    return gcd(aa, bb) - 1;
}
```

5.17 球面相关

```
//求大圆角度，返回大圆的圆心角，输入为弧度
double angle_3d(double lng1, double lat1, double lng2, double lat2)
```

```
{           //经度，纬度，经度，纬度
    return acos(cos(lat1)*cos(lat2)*cos(lng1 - lng2) +
sin(lat1)*sin(lat2));
}
//求大圆劣弧长度，输入为弧度
double dis_3d(double lng1, double lat1, double lng2, double lat2)
{           //经度，纬度，经度，纬度
    double dlon = lng2 - lng1;
    double dlat = lat2 - lat1;
     double a = pow((sin(dlat/2)),2) + cos(lat1) * cos(lat2) *
pow(sin(dlon/2), 2);
     double c = 2 * atan2(sqrt(a), sqrt(1-a));
     double d = r * c;
     return d;
}
//大地坐标转空间坐标系
//纬度， 经度 ，单位坐标， 输入为角度
    Point(double la, double lo) :
        x(cos(lo*PI/180) * cos(la*PI/180)),
        y(sin(lo*PI/180) * cos(la*PI/180)),
        z(sin(la*PI/180)) {}
```

5.18 模拟退火求多边形费马点

```
const int MAX = 1010;
const double inf = 1e30;
const double eps = 1e-8;
const double pi = acos(-1.0);
const int N = 15;    // 设定随机点的个数
const int L = 40;    // 设定随机方向次数
bool dy(double x,double y)    {    return x > y + eps;}    // x > y
bool xy(double x,double y)    {    return x < y - eps;}    // x < y
bool dyd(double x,double y)    {     return x > y - eps;}    // x >= y
bool xyd(double x,double y)    {    return x < y + eps;}     // x <= y
bool dd(double x,double y)     {    return fabs( x - y ) < eps;}  // x ==
y
struct point { double x,y;
    point (double x,double y):x(x),y(y){}
    point ():x(0),y(0){}
```

```
};
double disp2p(point a,point b) //  a b 两点之间的距离
{
    return sqrt(( a.x - b.x ) * ( a.x - b.x ) + ( a.y - b.y ) * ( a.y - b.y ));
}
point p[MAX];
point rp[MAX];
double len[MAX];
double min_dis(point a,point *p,int n)
{
    double min = inf;
    for(int i=0; i<n; i++)
    {
        double len = disp2p(a,p[i]);
        if( xy(len,min) )
            min = len;
    }
    return min;
}
bool check(point a,double x,double y)
{
    return dyd(a.x,0.0) && dyd(a.y,0) && xyd(a.x,x) && xyd(a.y,y);
}
point Rand(double x,double y)
{
    point c;
    c.x = ( rand()%1000 + 1 ) / 1000.0 * x;
    c.y = ( rand()%1000 + 1 ) / 1000.0 * y;
    return c;
}
int main()
{
    int n,m,ncases;
    double x,y;
    srand(time(NULL));     // time.h! !
    scanf("%d",&ncases);

    while( ncases-- )
    {
        scanf("%lf%lf%d",&x,&y,&n);
        for(int i=0; i<n; i++)
            scanf("%lf%lf",&p[i].x,&p[i].y);
        for(int i=0; i<N; i++)
        {
```

```
            rp[i] = Rand(x,y);
            len[i] = min_dis(rp[i],p,n);
        }
        point st;
        double step = max(x,y)/2;
        while( step > 0.001 )
        {
            for(int k=0; k<N; k++)
            {
                st = rp[k];
                for(int i=0; i<L; i++)
                {
                    double ang = (rand()%1000+1)/1000.0*10*pi;
                    double xx = st.x + step*cos(ang);
                    double yy = st.y + step*sin(ang);
                    point t = point(xx,yy);
                    if( !check(t,x,y) ) continue;
                    double dis = min_dis(t,p,n);
                    if( dy(dis,len[k]) )
                    {
                        rp[k] = t;
                        len[k] = dis;
                    }
                }
            }
            step *= 0.8;
        }
        int ind = 0;
        for(int i=1; i<N; i++)
            if( len[i] > len[ind] )
                ind = i;
        printf("The safest point is
(%.1lf, %.1lf).\n",rp[ind].x,rp[ind].y);
    }

return 0;
}
```

其他

6.1 BFS

BFS

```
void bfs(int n)
{
        memset(vd,0,sizeof(vd));
        queue< node >q;
        node in,out;
        in.v=n,in.w=1;
        vd[in.v]=1;q.push(in);\\入度标记，不能出队后标记
        while(!q.empty())
        {
                out=q.front(),q.pop();
                d[out.v]=out.w;
                for(int i=h[out.v];i>=0;i=e[i].next)
                {
                        if(vd[e[i].v])continue;
                        vd[e[i].v]=1;
                        in.v=e[i].v;in.w=out.w+1;
                        q.push(in);
                }
        }
}
```

6.2 表达式求值

表达式求值（加减乘除）

```
int r1,r2;
char q2[10000];
double q1[10000];
double opt(double a,double ch,double b)
{
    if(ch=='+')return a+b;
    if(ch=='-')return a-b;
    if(ch=='*')return a*b;
    if(ch=='/')return a/b;
}
double get_num(char s[])
{
    r1=r2=0;
    double in,out;
    int len=strlen(s);
```

```
    for(int i=0;i<len;i++)
    {
        if(s[i]=='+'||s[i]=='-'||s[i]=='*'||s[i]=='/'||s[i]=='(')
            q2[r2++]=s[i];
        else if(s[i]<='9'&&s[i]>='0')
        {
            in=0;
            while(s[i]<='9'&&s[i]>='0')
            {
                in=in*10+s[i]-'0';
                i++;
            }
            i--;
            if(q2[r2-1]=='*'||q2[r2-1]=='/')
            {
                out=q1[--r1];
                in=opt(out,q2[--r2],in);
                q1[r1++]=in;
            }
            else q1[r1++]=in;
        }
        else if(s[i]==')')
        {
            int k=r2-1,l=0;
            while(q2[k]!='(')
                k--,l++;
            r2=k;in=q1[r1-l-1];
            for(int i=r1-l;i<r1;i++)
                in=opt(in,q2[++k],q1[i]);
            r1-=l+1;
            if(q2[r2-1]=='*'||q2[r2-1]=='/')
            {
                out=q1[--r1];
                in=opt(out,q2[--r2],in);
                q1[r1++]=in;
            }
            else q1[r1++]=in;
        }
    }
    in=q1[0];
    for(int i=0;i<r2;i++)
        in=opt(in,q2[i],q1[i+1]);
    return in;
}
```

```
int main()
{
    char s[10005];
    while(1)
    {
        gets(s);
        if(strlen(s)==1&&s[0]=='0')
            break;
        printf("%.2f\n",get_num(s));
    }
    return 0;
}
```

6.3 并查集

并查集

```
int Find(int x)
{
        return x==fa[x]?x:fa[x]=Find(fa[x]);//并查集精华，压缩路径
}
void unin(int x,int y)//还可以按深度合并，加多个深度标记即可
{
        int a=Find(x);
        int b=Find(y);
        fa[a]=b;
}
void init(int n)//初始化时自己为自己的父亲
{
        for(int i=0;i<=n;i++)
                fa[i]=i;
}
```

注意关系的传递，回溯时对关系的更新，并查集判连通快速，准确。

带权值的更新与转移

```
int sum[200005],fa[200005];
int find(int n) //推出 n 于 fa[n]的关系然后更新
{
        if(n!=fa[n])
        {
           int x=find(fa[n]);
           sum[n]+=sum[fa[n]];
           fa[n]=x;
        }
        /*else return n;*/
```

```
        return fa[n];
}
void unin(int x,int y,int w)//推出 x 于 y 祖先的关系然后合并
{
        int a=find(x),b=find(y);
        fa[a]=b;
        sum[a]=sum[y]-sum[x]+w;
}
void init(int n)
{
        for(int i=0;i<=n;i++)
                fa[i]=i,sum[i]=0;
}
```

6.4 博弈

基础博弈

（一） **巴什博奕（Bash Game）**：只有一堆 n 个物品，两个人轮流从这堆物品中取物，规定每次至少取一个，最多取 m 个。最后取光者得胜。

解法：找出必败点即可，所有可以转移到必败点的都是必胜点。

必败点：无论怎么操作都只能到达必胜点。

（二） **威佐夫博奕（Wythoff Game）**：有两堆各若干个物品，两个人轮流从某一堆或同时从两堆中取同样多的物品，规定每次至少取一个，多者不限，最后取光者得胜。

解法：可以画表格 x,y 表示两堆的数量，找出必胜和必败点。

奇异局面：谁处在奇异局面，谁必败。

规律： ak =[k（1+√5）/2]，bk= ak + k （k=0，1，2，…,n 方括号表示取整函数)

只需先求出 k 然后判断 ak =[k（1+√5）/2]这个式子是否成立，成立后再判 bk= ak + k 是否成立，都成立既为奇异局势。

（三） **尼姆博奕（Nimm Game）**：有三堆各若干个物品，两个人轮流从某一堆取任意多的物品，规定每次至少取一个，多者不限，最后取光者得胜。

解法：各个堆的值异或即可。

引入：Anti-SG (SJ 定理)

SJ 定理：

对于任意的一个 Anti-SG 游戏，如果我们规定当局面中所有单一游戏的 SG 值为 0 时游戏结束，则先手必胜当且仅当以下两个条件满足任意一个：

(1)游戏的 SG 函数不为 0，且游戏中某个单一游戏的 SG 函数大于 1。

(2)游戏的 SG 函数为 0，且游戏中没有单一游戏的 SG 数大于 1。

组合博弈：由多个子问题组成。

解法：对每个子问题求 sg 值，然后全部异或即可。

SG 值：一个点的 SG 值就是一个不等于它的后继点的 SG 值且大于等于零的最小整数。例如 mex{0,1,2,4}=3、mex{2,3,5}=0、mex{}=0。

SG 更简单的描述：

在步骤允许的情况下，与前面一个必败点的差（也就是说这个差是规定的、能走的、其中一个改变的值）！

```
int SG(int n)
{
        if(sg[n]!=-1)return sg[n];
        bool vd[1005]={0};
        for(int i=0;i<len&&n-arry[i]>=0;i++)//arry 数组表示题目要求可以操作的数量
        {
                int tem=n-arry[i];
                sg[tem]=SG(tem);
                vd[sg[tem]]=1;
        }
        for(int i=0;1;i++)
                if(!vd[i])
                        return sg[n]=i;
}
```

6.5 矩阵

一般考虑方法

（一）Fibonacci 数列 f[n]=f[n-1]+f[n-2],f[1]=f[2]=1 的第 n 项的快速求法（不考虑高精度）.

解法：

考虑 1×2 的矩阵【f[n-2],f[n-1]】。根据 fibonacci 数列的递推关系，我们希望通过乘以一个 2×2 的矩阵，得到矩阵【f[n-1],f[n]】=【f[n-1],f[n-1]+f[n-2]】

很容易构造出这个 2×2 矩阵 A，即：

0	1
1	1

所以，有【f[1],f[2]】×A=【f[2],f[3]】

数列 f[n]=f[n-1]+f[n-2]+n+1,f[1]=f[2]=1 的第 n 项的快速求法（不考虑高精度）.

解法：

仿照前例，考虑 1×4 的矩阵【f[n-2],f[n-1],n,1】，希望求得某 4×4 的矩阵 A，使得此 1×4 的矩阵乘以 A 得到矩阵：

【f[n-1],f[n],n+1,1】=【f[n-1],f[n-1]+f[n-2]+n+1,n+1,1】

容易构造出这个 4×4 的矩阵 A，即：

0	1	0	0
1	1	0	0
0	1	1	0
0	1	1	1

问题（三）解决……

递推矩阵构造时，矩阵的项能够从之前的项转移过来

数列 **f[n]=f[n-1]+f[n-2],f[1]=f[2]=1** 的前 **n** 项和 **s[n]**的快速求法（不考虑高精度）.

解法：

虽然我们有 **S[n]=F[n+2]-1**，但本文不考虑此方法，我们想要得到更一般的方法。

考虑（一）的矩阵 **A**，容易发现我们要求【**f[1],f[2]**】×（$A+A^2+A^3+...+A^{N-1}$）。很多人使用一种很数学的方法构造一个 **2r*2r**（**r** 是 **A** 的阶数，这里为 **2**）的矩阵来计算，这种方法比较麻烦且很慢，这里不再介绍。下面考虑一种新方法。

仿照之前的思路，考虑 **1×3** 的矩阵【**f[n-2],f[n-1],s[n-2]**】，我们希望通过乘以一个 **3×3** 的矩阵 **A**，得到 **1×3** 的矩阵：

【f[n-1],f[n],s[n-1]】=【f[n-1],f[n-1]+f[n-2],s[n-2]+f[n-1]】

一般地，如果有 **f[n]=p*f[n-1]+q*f[n-2]+r*n+s**

可以构造矩阵 **A** 为：

0	q	0	0	0
1	p	1	0	0
0	0	1	0	0
0	r	0	1	0
0	s	0	1	1

更一般的，对于 **f[n]=Sigma(a[n-i]*f[n-i])+Poly(n)**，其中 **0<i<=**某常数 **c, Poly (n)** 表示 **n** 的多项式，我们依然可以构造类似的矩阵 **A** 来解决问题。

设 **Degree(Poly(n))=d,** 并规定 **Poly(n)=0** 时，**d=-1**，此时对应于常系数线性齐次递推关系。则本方法求前 **n** 项和的复杂度为：

$((c+1)+(d+1))^3*\log n$

例题

g(i)=k*i+b;（等差数列）

f(0)=0

f(1)=1

f(n)=f(n-1)+f(n-2) (n>=2)

计算 sum= f(g(i)) for 0<=i<n

构造矩阵 A

0 1

1 1

设 F[n]=A^n(F[n]为最终矩阵左下角的值)

Sum=f(b)+f(k+b)+f(2*k+b)+f(3*k+b)+…..f((n-1)*k+b)

=A^b+A^(k+b)+A^(2*k+b)……+A^((n-1)*k+b)

=A^b+A^b(A^k+A^2k+….A^(n-1)k)

设 B=A^k

Sum=A^b+A^b(B^1+B^2+B^3+…..B^n-1)

B^1+ B^2+ B^3+ B^4+ B^5=B^1+B^2+B^3+B^3(B^1+B^2)(这里可以二分快速求和)

```
#define LG long long
using namespace std;
const int maxn=10005,MAX=1<<30,MIN=-(1<<30);
LG mod,inc,n,m;
struct node
{
        LG gp[5][5];
}A,B;
node mul(node a,node b)//矩阵相乘
{
        node c;
        for(int i=0;i<2;i++)
                for(int j=0;j<2;j++)
                {
                        c.gp[i][j]=0;
                        for(int k=0;k<2;k++)
                        {
                                c.gp[i][j]+=a.gp[i][k]*b.gp[k][j];
                                c.gp[i][j]%=mod;
                        }
                }
                return c;
}
node sort_exp(LG k,node norm)//矩阵快速幂 norm^k
{
        if(k==0)return B;
        if(k==1)return norm;
        node a,b;
        b=sort_exp(k/2,norm);
        a=mul(b,b);
        if(k&1)a=mul(a,norm);
        return a;
}
node add(node a,node b)//矩阵相加
{
        node c;memset(c.gp,0,sizeof(c.gp));
        for(int i=0;i<2;i++)
                for(int j=0;j<2;j++)
                        c.gp[i][j]=(a.gp[i][j]+b.gp[i][j])%mod;
```

```
        return c;
}
node sort_sum(LG k,node norm)//矩阵二分求和 norm^1+norm^2+……+norm^k
{
        if(k==0)return B;
        if(k==1)return norm;
        node a,b,c,d;
        b=sort_sum(k/2,norm);
        if(k&1)
        {
                c=sort_exp(k/2+1,norm);
                a=mul(c,b);
                d=add(add(a,b),c);
        }
        else
        {
                c=sort_exp(k/2,norm);
                a=mul(c,b);
                d=add(a,b);
        }
        return d;
}
int main()
{
        A.gp[0][0]=0;A.gp[0][1]=A.gp[1][0]=A.gp[1][1]=1;
        for(int i=0;i<2;i++)B.gp[i][i]=1;
        while(~scanf("%d%d%d%d",&inc,&m,&n,&mod))//题中的 k,b,n,mod;
        {
                node A_b,A_k,ans;
                A_b=sort_exp(m,A);A_k=sort_exp(inc,A);
                ans=sort_sum(n-1,A_k);
                ans=mul(ans,A_b);
                ans=add(ans,A_b);
                cout<<ans.gp[1][0]<<endl;
        }
        return 0;
}
```

非递归写法

```
node sort_exp(int k)
{
        node ans=B,a=A;
        while(k)
        {
                if(k&1)ans=mul(ans,a);
```

```
                k>>=1;
                a=mul(a,a);
        }
        return ans;
}
```

6.6 二分搜索

二分搜索

对于单调函数的二分求最值

```
double r=Max,l=Min,mid;
while(fabs(r-l)>=1e-4)//一般精确到所要保留的小数的后两位
{
    mid=(r+l)/2;
    if(isok(mid))
        l=mid;
    else
        r=mid;
}
printf("%.2f\n",(l+r)/2);//结果是(l+r)/2;
```

三分求极值

```
double r=Max,l=Min,mid,mid2;
while(fabs(r-l)>=1e-5) //一般精确到所要保留的小数的后两位
{
      mid=(r+l)/2;
      mid2=(mid+r)/2;
      if(vm(mid)<vm(mid2))
          r=mid2;
      else
          l=mid;
}
printf("%.3f\n",(r+l)/2.0);
```

二分搜索

```
int serch(int k,int Min,int Max)
{
        int mid,l=Min,r=Max;
        while(l<=r)
        {
                mid=(l+r)>>1;
                if(a[mid]==k)return mid;//找到返回该数的下标
                else if(a[mid]<k) l=mid+1;
```

```
                else r=mid-1;
        }
        return l;//没找到返回比 k 大的第一个数
}
```

6.7 归并排序&逆序对&分治三步曲

```
void merge(double *a,int x,int y,double *b)
{
        if(y-x<=1)return;
        int mid=(x+y)>>1;
        int pos=x,p=x,q=mid;
        merge(a,x,mid,b);
        merge(a,mid,y,b);
        while(p<mid||q<y)
        {
                if(q>=y||(p<mid&&a[p]<=a[q]))
                        b[pos++]=a[p++];
                else b[pos++]=a[q++],cnt+=mid-p;//cnt 保存逆序对数
        }
        for(int i=x;i<y;i++)
                a[i]=b[i];
}
```

6.8 母函数

硬币是 1、2、5。（每一种的母函数相乘就是结果）
所以可以写成：
（1 + X + X^2 + .. X^num_1）(1 + X^2 + X^4 + X^6 + ... X^num_2*2)(1 + X^5 + X^10 + X^15 + ... X^num_3*5)
其中三个括号元素的个数分别是输入的 num_1，num_2，num_3。然后就是模拟多项式相乘了。*/结果中最大的次方数为 X^(num_1)+ X^(num_2*2)+ X^(num_3*5)
最后结果 X 的指数 i 表示系数是组合数 C(num_1+num_2+num_3,i);
生成函数 g(x)=1+x+x^2+x^3+x^4+...（每一项都是一，即使 n=0 时也有 x^0 系数为 1，所以有常数项）。再仔细一看，这就是一个有无穷多项的等比数列求和嘛。如果-1<x<1，那么 g(x)就等于 1/(1-x)了。
N 个 g(x)相乘就是：
1/(1-x)^n=1+C(n,1)x^1+C(n+1,2)x^2+C(n+2,3)x^3+...+C(n+k-1,k)x^k+.....
整数划分

```
#define LG long long
using namespace std;
const int maxn=10005,MAX=1<<30,MIN=-(1<<30);
```

```
LG dp[maxn],dp1[maxn]; int n;
LG DP(int n)
{
        for(int i=0;i<=n;i++)
                dp[i]=1,dp1[i]=0;
        for(int i=2;i<=n;i++)
        {
                for(int j=0;j<=n;j++)
                        for(int k=0;j+k<=n;k+=i)
                                dp1[k+j]+=dp[j];
                for(int j=0;j<=n;j++)
                        dp[j]=dp1[j],dp1[j]=0;
        }
        return dp[n];
}
int main()
{
        while(~scanf("%d",&n))
        {
                cout<<DP(n)<<endl;
        }
}
```

6.8 KMP

```
#define MAX 1001
using namespace std;

int next[MAX];
char s[MAX],t[MAX];

void get_next(char *s)
{
    int len = strlen(s);
    int i = 0,j = -1;
    next[0] = -1;
    while( i < len )
        if( j == -1 || s[i] == s[j] )
        {
            i++, j++;
            next[i] = s[i] == s[j] ? next[j] : j;
```

```
        }
        else
            j = next[j];
}
int KMP(char *s,char *t)
{
    get_next(t);
    int lens = strlen(s);
    int lent = strlen(t);
    int i = 0,j = 0;
    while( i < lens && j < lent )
        if( j == -1 || s[i] == t[j] )
            i++, j++;
        else
            j = next[j];
    if( j >= lent )
        return i - lent;
    return -1;
}
int main()
{
    int n,pos;
    while( scanf("%s %s",&s,&t)!=EOF )
    {
        pos = KMP(s,t);
        if( pos == -1 )
            printf("Can't match\n");
        else
            printf("%d\n",pos);
    }
return 0;
}
```

KMP 寻找循环节

如果 len%(len-next[len])==0,则字符串中必存在最小循环节，且循环次数即为 len/(len-next[len])（若要求循环次数不等于 1，则 next[len]!=0）
要注意的是 len - next[len]为这个串的最小循环节的长度，这不需要最后的子串是完整的； if(len % (len - next[len]) == 0)这个求得的是串的最大的循环的个数这个要求最后的子串是完整的
且函数修改如下

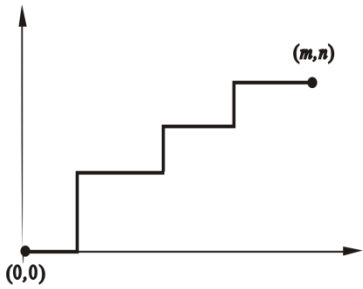
```
void get_next(char *s)
{
    int len = strlen(s);
    int i = 0,j = -1;
```

```
    next[0] = -1;
    while( i < len )
        if( j == -1 || s[i] == s[j] )
        {
            i++, j++;
            next[i] = j;
        }
        else
            j = next[j];
}
```

6.9 组合数学（yuan）

非降路径问题

1. (0,0) 到 (*m,n*) 的非降路径数 *C*(*m+n*, *m*)



总共 m+n 步，选择其中的 m 步向上

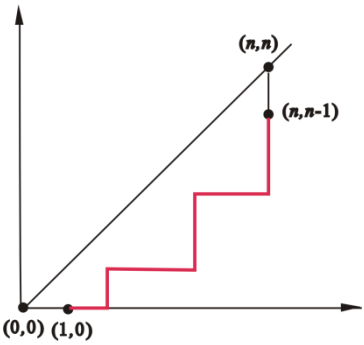
而两点间的路径数就是 C(x2-x1+y2-y1,x2-x1).

变形

限制：**中间有一些不能走的点** （SCAU2010 校赛 G 题）

用**容斥**即可　**|S| - |P1∪...∪Pn|**　Pi 表示必须经过第 i 个性质

2. 从(0,0)到(n,n)不接触对角线的非降路径数 N



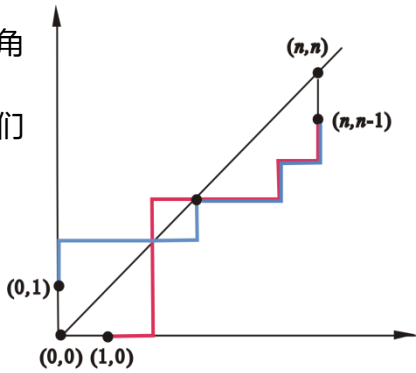
也即从(1,0)到(n,n-1)不接触对角线的路径数，为

(1,0)到(n,n-1)的所有路径数 - (1,0)到(n,n-1)接触对角线的路径数

(1,0)到(n,n-1)接触对角线的路径数：

就是(0,1)到(n,n-1)的路径数

说明：

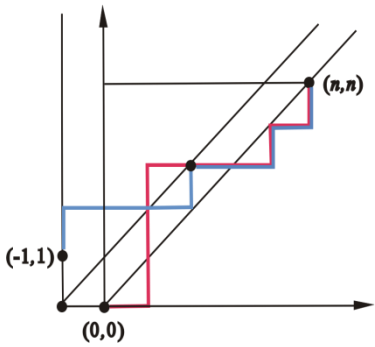


假设**(0,1)到(n,n-1)**接触对角线上的 **A 点**，将(0,1)到 A 的路径沿对角线对折，变成(1,0)到 A ，这条从(1,0)到(n,n-1)的路劲必接触对角线，它们是一一

映射关系。或者说，**任何一条从(1,0)到(n,n-1)接触对角线的路径必能找到一条从(0,1)到(n,n-1)的路径**

答案： C(2n-2,n-1) – C(2n-2,n)

3.不穿过对角线（可接触）路径数　　进栈出栈方案数



任何前缀出栈的次数肯定≤进栈次数，也即 y 方向的步数≤x 方向的步数

跟上面的类似，用**总的减去穿过的**

这次，(0,0)对折到(-1,1)

答案为 C(2n,n) – C(2n,n-1)

Hdu 3398 求一个 n 个 1,m 个 0 组成的 01 串，任何前缀中 0 的个数要≤1 的个数的串的总数。直接用上面的公式了，注意这题求 N！不能直接 O(n)，需要分解质因子，利用

N!中质因子 p 的幂为 N / p +N/ p^2 + ...

```
/*
    n*n的棋盘，m种颜色的棋子，每种颜色的棋子有num[i]颗
    要将这些棋子放到棋盘去，一个位置只能放一颗
    求方案数目，若旋转后一样的话，算一种方案
    n , num[i]<= 100000, m <= 50
```

```
    要用到Burnside引理：
    黑书P250
    "如果记C(f)为在置换f下保持不变的着色方案个数，则本质不同的着色方案数为所有置换f
的
    C(f)值的平均数"
-----------------------------------------------OMG

    这道题就是[C(0°)+C(90°)+C(180°)+C(270°)]/ 4，里面的数字表示旋转的角度

    题目的棋盘是正方形，比较特殊，方便计数
    C(0°)的比较容易计算
    答案即为C(n^2, num[1]) * C(n^2 - num[1], num[2]) * ...

    C(90°)，表明该正方形的个小正方形是旋转一样（奇数的中间那一个不用考虑）
    因此只要确定了其中一个小正方形就可以确定方案数目了
    即为C(n^2/4, num[1]/4) * C(n^2/4 - num[1]/4, num[2]/4) * ...  （这里只考虑
偶数的，奇数类似）

    C(180°)，表明该正方形关于中心点旋转对称
    因此只要确定了正方形上面一半就可以确定方案数目了
    即为C(n^2/2, num[1]/2) * C(n^2/2 - num[1]/2, num[2]/2) * ...  （这里只考虑
偶数的，奇数类似）

    注意的是不要对公式
    C(n^2, num[1]) * C(n^2 - num[1], num[2]) * ...
    化简为
    n^2!/[num[1]!*num[2]!*....(n^2-num[1]-num[2]...)!]
    这样难算的！！n^2为^10
    我之前还以为能用分解质因子，sb了，^10那么大的素数哪里来？....
    直接算就行了50*10^5

    1234567891是素数，逆元方便求
    1234567891*1234567891本来是超long long的，但这题没有这样的数据

    这题时限s
    跑了s多

    数据
    http://www.cppblog.com/Files/Yuan/D_Deploy%20the%20Checker.rar
*/
const long long MOD = 1234567891LL;
const int MAXN = 100100;

long long inv[MAXN];
```

```
long long ipow(long long p, long long n)
{
    if (n == 0 ) return 1;
    if (n == 1) return p % MOD;
    long long ans = ipow(p, n/2);
    ans = ans * ans % MOD;
    if (n&1) ans = ans * p % MOD;
    return ans;
}

long long C(long long n, int m)
{
    long long ans = 1;
    for(int i = 1;i <= m; i++) {
        ans = (n-i+1) % MOD * ans % MOD;
        ans = ans * inv[i] % MOD;
    }
    return ans;
}

long long gao(long long n, vector<int> &num)
{
    long long ans = 1;
    for (vector<int>::iterator it = num.begin(); it != num.end(); it++) {
        ans = ans * C(n, *it) % MOD;//
        n -= *it;
    }
    return ans;
}

int main()
{
    //long long start = clock();
    for (int i = 1; i < MAXN; i++){
        inv[i] = ipow(i, MOD-2) % MOD;
    }
    long long n;
    int m;
    while(cin>>n>>m) {
        vector<int> num;
        for(int i = 0, x; i < m; i++) {
            scanf("%d", &x);
            num.push_back(x);
```

```
        }
        n *= n;
        if (accumulate(num.begin(), num.end(), 0LL) > n) {
            puts("0");
            continue;
        }
        long long ans = gao(n,num);
        if (n&1) {
            vector<int> _num(num);
            bool error = false;
            int cnt = 0;
            for (vector<int> ::iterator it = _num.begin(); it != _num.end(); it++)
{
                if (*it & 1) {
                    cnt++;
                    (*it)--;
                }
                if (*it % 4 != 0) {
                    error = true;
                    break;
                }
                *it /= 4;
            }
            if (!error && cnt <= 1 ) {
                ans = (ans + 2 * gao((n-1)/4, _num)) % MOD;
            }
            _num = num;
            error = false;
            cnt = 0;
            for (vector<int> ::iterator it = _num.begin(); it != _num.end(); it++)
{
                if (*it & 1) {
                    cnt++;
                    (*it)--;
                }
                if (*it % 2 != 0) {
                    error = true;
                    break;
                }
                *it /= 2;
            }
            if (!error && cnt <= 1 ) {
                ans = (ans + gao((n-1)/2, _num)) % MOD;
            }
```

```
        } else {
            vector<int> _num(num);
            bool error = false;
            for (vector<int> ::iterator it = _num.begin(); it != _num.end(); it++)
{
                if (*it % 4 != 0) {
                    error = true;
                    break;
                }
                *it /= 4;
            }
            if (!error) {
                ans = (ans + 2 * gao(n/4, _num)) % MOD;
            }
            _num = num;
            error = false;
            for (vector<int> ::iterator it = _num.begin(); it != _num.end(); it++)
{
                if (*it % 2 != 0) {
                    error = true;
                    break;
                }
                *it /= 2;
            }
            if (!error) {
                ans = (ans + gao(n/2, _num)) % MOD;
            }
        }
        cout<< ans * inv[4] % MOD <<endl;
    }
    //fprintf(stderr, "%lld", clock() - start);
    return 0;
}
```

DP

7.1 金典模型(参考 yuan 神资料)

LIS nlogn

主要看那个二分就行了还有 rec[]的含义

Zoj 2319

```
/*
    二维LIS，可以先按s排序，这样就可以不用管s了（注意相等时按b>b`排）
    要nlogn才能过
    参考poj1609
    哎，居然忘记了。。
    要加那个pre[]输出答案pre[i]=rec[low-1]
    rec[len]表示长度为len的最小元素
*/
#include<cstdio>
#include<cstring>
#include<algorithm>
usingnamespace std;

constint MAXN=100010;

struct Peo
{
    int s,b,id;
    booloperator<(const Peo &p)const{
        if(s==p.s)return b>p.b;//相等时，这样为了满足 s<=s`&&b<=b`，令之大于的话就不会破坏条件
        return s<p.s;
    }
}peo[MAXN];

int rec[MAXN],pre[MAXN];

int main()
{
    int T,n;
    scanf("%d",&T);
    while(T--){
        scanf("%d",&n);
        for(int i=1;i<=n;i++){
            scanf("%d%d",&peo[i].s,&peo[i].b);
            peo[i].id=i;
        }
        sort(peo+1,peo+1+n);
        //排完序后，就可以不用管 s
        int len=0;
        for(int i=1;i<=n;i++){
            int low=1,high=len+1;
            while(high>low){
```

```
                int mid=(high+low)>>1;
                if(peo[rec[mid]].b<peo[i].b)low=mid+1;
                else high=mid;
            }
            if(low==len+1)len++;
            rec[low]=i;
            pre[i]=rec[low-1];//保存 i 的 pre!
        }
        printf("%d\n",len);
        printf("%d",peo[rec[len]].id);
        int x=pre[rec[len]];
        while(x){
            printf(" %d",peo[x].id);
            x=pre[x];
        }
        puts("");
    }
    return 0;
}
```

最长公共上升子序列

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

int dp[510];
int s[510][510];
int A[510],B[510];
int n,m,cnt;
bool flag;

void print(int i,int j)
{
    if(i==0||j==0)return;
    if(A[i]==B[j])
    {
        print(i-1,s[i][j]);
        if(flag)putchar(' ');
        else flag=true;
        printf("%d",A[i]);
    }
```

```
    else print(i-1,j);
}

int main()
{
    int T;
    scanf("%d",&T);
    while(T--)
    {
        scanf("%d",&n);
        for(int i=1;i<=n;i++)
            scanf("%d",&A[i]);
        scanf("%d",&m);
        for(int i=1;i<=m;i++)
            scanf("%d",&B[i]);
        memset(dp,0,sizeof(dp));
        int Max = 0,ii=0,jj=0;
        for(int i=1;i<=n;i++)
        {
            int loc = 0;
            for(int j=1;j<=m;j++)//还没更新时,dp[j]即dp[i-1,j]表示的是前i-1
个阶段的最优
            {
                if(A[i]>B[j]&&dp[j]>dp[loc])loc=j;
                if(A[i]==B[j])
                {
                    dp[j]=dp[loc]+1;//dp[i,j] = dp[k,loc]+1
                    s[i][j]=loc;
                    if(dp[j]>Max)Max=dp[j],ii=i,jj=j;
                }
            }
        }
        printf("%d\n",Max);
        flag = false;
        print(ii,jj);
        puts("");
    }
    return 0;
}
```

Poj 1191★★棋盘划分

```
/*
"此题乍看貌似是搜索题，由于棋盘大小固定是行列。枚举所有的切割可能性。。然后发现以我的水平还无法去实现。。
然后发现题目中一句话说的切割必须要沿着棋盘格子的边进行。貌似 dp。。。"

黑书 P116
均方差公式比较复杂，先将其变形
S^2 = 1/n∑(Xi - X)^2
    = 1/n(n*X^2 + ∑Xi^2 - 2*X∑Xi)
    = 1/n∑Xi^2 - X^2;
    X 表示均值
    由于均值是一定的（它等于方格里的数和除以 n），所以只需要让每个矩形的总分的平方和尽量小
    对于左上角(x1,y1)，右下角(x2,y2)的正方形，设它的总和为 S[x1,y1,x2,y2]
    切割 K 次后得到的 K+1 块矩形的总分平方和最小值为 dp[k,x1,y1,x2,y2]
    则它可以横着切，也可以竖着切，然后选一块切（这里用到了递归！）
    dp[k,x1,y1,x2,y2] = min{
        min{ dp[k-1,x1,y1,a,y2]+S[a+1,y1,x2,y2] ,
dp[k-1,a+1,y1,x2,y2]+S[x1,y1,a,y2] }(x1<=a<x2),
        min{ dp[k-1,x1,y1,x2,b]+S[x1,b+1,x2,y2] ,
dp[k-1,x1,b+1,x2,y2]+S[x1,y1,x2,b] }(y1<=b<y2)
    }
    设 m 为棋盘边长，则状态数目为 m^4n，决策数目为 O(m)
    O(m^5n))
*/
#include<iostream>
#include<cstring>
#include<cmath>
#include<cstdio>

usingnamespace std;

int dp[16][9][9][9][9];
int Map[9][9];
int S[9][9];

void init(){
    for(int i=1;i<=8;i++)
        for(int j=1;j<=8;j++)
            S[i][j] = S[i-1][j] + S[i][j-1] - S[i-1][j-1] + Map[i][j];
}
```

```
int calc(int x1,int y1,int x2,int y2){
    //S[y][x]
    int tmp = S[y2][x2] - S[y2][x1-1] - S[y1-1][x2] +S[y1-1][x1-1];
    return tmp*tmp;
}

int Dp(int N,int x1,int y1,int x2,int y2){
    if(dp[N][x1][y1][x2][y2]>=0)
        return dp[N][x1][y1][x2][y2];
    if(N>1){
        int Min = 1073741824;//1<<30
        for(int i = x1;i<x2;i++){
            int tmp;
            tmp = Dp(N-1,x1,y1,i,y2)+calc(i+1,y1,x2,y2);
            Min = min(Min,tmp);
            tmp = Dp(N-1,i+1,y1,x2,y2)+calc(x1,y1,i,y2);
            Min = min(Min,tmp);
        }
        for(int i = y1;i<y2;i++){
            int tmp;
            tmp = Dp(N-1,x1,y1,x2,i)+calc(x1,i+1,x2,y2);
            Min = min(Min,tmp);
            tmp = Dp(N-1,x1,i+1,x2,y2)+calc(x1,y1,x2,i);
            Min = min(Min,tmp);
        }
        dp[N][x1][y1][x2][y2] = Min;
        //像 dp[N][x1][y1][x1][y2] 或 dp[N][x1][y1][x2][y1]就会=
1073741824;
        return Min;
    }
    else {
        dp[1][x1][y1][x2][y2] = calc(x1,y1,x2,y2);
        return dp[1][x1][y1][x2][y2];
    }
}

int main(){
    int N;
    double sum = 0;
    cin>>N;
    memset(dp,-1,sizeof(dp));//memset(dp,128,sizeof(dp));
    for(int i=1;i<=8;i++)
        for(int j=1;j<=8;j++){
            cin>>Map[i][j];
```

```
            sum+=Map[i][j];
        }

    init();

    sum/=N;
    double Avg = (double)Dp(N,1,1,8,8);
    printf("%.3f\n",sqrt(Avg/N - sum*sum));

   return 0;
}
```

```
/*
分析线性 DP

参照《算法艺术》上的算法

题目的方块可以表示称 color[i],len[i],1<=i<=l
这里 l 表示有多少"段"不同的颜色方块
    color[i]表示第 i 段的颜色,len[i]表示第 i 段的方块长度
让 f[i,j,k]表示把
(color[i],len[i]),(color[i+1],len[i+1]),...,(color[j-1],len[j-1]),(colo
r[j],len[j]+k)合并的最大得分
这里 k 是可跟 j 合并的长度
考虑(color[j],len[j]+k)这一段,要不马上消掉,要不和前面的若干段一起消掉
.............................................................
    1.如果马上消掉,就是 f[i,j-1,0]+(len[j]+k)^2
    2.如果和前面的若干段一起消,可以假设这"若干段"中最后一段是 p,则此时的得分是
f[i,p,k+len[j]]+f[p+1,j-1,0]

于是 f[i,j,k] = max{ f[i,j-1,0]+(len[j]+k)^2,
f[i,p,k+len[j]]+f[p+1,j-1,0] }
    其中 color[p]=color[j] ,i<=p<j 边界条件 f[i,i-1,0]=0;


复杂度

    O(n^4),时间效率很低,但可以利用更新解的必要条件 color[p] = color[j]来优化
算法
```

```
*/
constint MAXN = 201;
int dp[MAXN][MAXN][MAXN],len[MAXN],color[MAXN];
int DP(int i,int j,int k){
    if(dp[i][j][k]!=-1)return dp[i][j][k];
    if(i==j+1&&k==0)return 0;
    dp[i][j][k] = DP(i,j-1,0)+(len[j]+k)*(len[j]+k);
    for(int p=j-1;p>=i;p--)
        if(color[p]==color[j]){
            int tmp = DP(i,p,k+len[j])+DP(p+1,j-1,0);
            if(dp[i][j][k]<tmp)dp[i][j][k]=tmp;
        }
    return dp[i][j][k];
}
int main(){
    int T,n,colorNum,tmp;
    cin>>T;
    for(int t=1;t<=T;t++){
        cin>>n>>color[colorNum=1];
        len[colorNum]=1;
        for(int i=1;i<n;i++){
            cin>>tmp;
            if(tmp!=color[colorNum]){//输入的处理
                color[++colorNum] = tmp;
                len[colorNum]=1;
            }
            else len[colorNum]++;
        }
        memset(dp,-1,sizeof(dp));
        cout<<"Case "<<t<<": "<<DP(1,colorNum,0)<<endl;
    }
    return 0;
}
```

双调 tsp

Noip 2007 特种部队转移是枚举哪一个去接 max(i,j)+1

```
/*
```

```
    一排按钮，要求从第一个，按到最后一个，再从最后一个按回来
    中间可以跳着按，但要求每个按钮有且按一次，求使得相邻按的按钮的分数差之和最小
    给出 n<=1000 个数

    参考
    http://hi.baidu.com/xiszishu/blog/item/650f5b8e094b07fef11f3667.html

    双进程 dp   双调 tsp
    由于最后不会按回，所以加多一个按钮，到各个按钮的差异为
    dp[i,j] i < j
    表示从一个从按到 i,一个从按到 j 的最小代价
    转移是枚举 i,j 哪个下一步按 max(i,j)+1 （扩展一步去转移丫，我就不会这个转移了！！！）
    dp[i,k] = max(dp[i,k] + dp[i,j] + abs(a[j]-a[k]))
    dp[k,j] = max(dp[k,j] + dp[i,j] + abs(a[i]-a[k]))
    答案就是 dp[n-1,n] + abs(a[n-1]-a[n])

*/
constint MAXN = 1010;
constint INF = 1000000000;

int a[MAXN];
int dp[MAXN][MAXN];

int main()
{
#ifndef ONLINE_JUDGE
    freopen("in","r",stdin);
#endif

    for(int n; ~scanf("%d", &n); ){
        for(int i = 1 ; i <= n ; i++){
            scanf("%d", &a[i]);
        }
        fill(dp[0], dp[MAXN], INF);
        dp[0][0] = 0;
        for(int i = 0; i < n; i++){
            for(int j = i; j < n ; j++){//j = i  因为从[0,0]开始
                if(dp[i][j] >= INF){
                    continue;
                }
                int k = j + 1;
```

```
                dp[i][k] = min(dp[i][k], dp[i][j] + (j == 0 ? 0 : abs(a[j]
- a[k])) );
                dp[j][k] = min(dp[j][k] , dp[i][j] + (i == 0 ? 0 : abs(a[i]
- a[k])));
            }
        }
        printf("%d\n", dp[n-1][n] + abs(a[n]-a[n-1]));
    }
    return 0;
}
```

树 DP

给你 **m** 个机器人从 **s** 点出发，，每条路由一定的权值，求历遍整棵树最小的权值。

显然是树 **dp**，主要难点是那个类似背包的递推，其实是这样的，每一颗子树相当于一个物品 **i**，但是容量 **k**（机器人数）可以任选，将它放进总容量中，使得这个总容量 **j** 的权值最优，然后由第一颗子树推到最后，跟背包一样，由前 **i** 个物品的最优和当前的物品匹配选最优，我们知道 **01** 背包能化为 **1** 维的，这里也是这样优化的。。

```
#define LL long long
using namespace std;
struct egde
{
    int v,w,next;
}e[20020];
int fa[10020],n,s,m,pos;
LL dp[10020][12];
void DP(int t,int p)
{
    for(int i=fa[t];i!=-1;i=e[i].next)
    {
        if(e[i].v==p)continue;
        DP(e[i].v,t);
    }
    for(int i=fa[t];i!=-1;i=e[i].next)
```

```
    {
        if(e[i].v==p)continue;
        int tem=e[i].v;
        for(int j=10;j>=0;j--)
        {

            if(dp[t][j]==0)
            {
                if(j==0)dp[t][0]=dp[tem][0]+2*e[i].w;
                else
                    for(int k=1;k<=j;k++)
                    {
                        if(dp[t][j]==0||dp[t][j]>dp[tem][k]+k*e[i].w)
                            dp[t][j]=dp[tem][k]+k*e[i].w;
                    }
                    //continue;
            }
            else
            {
                dp[t][j]+=2*e[i].w+dp[tem][0];
                for(int k=1;k<=j;k++)
                {
                   dp[t][j]=min(dp[t][j],dp[t][j-k]+dp[tem][k]+k*e[i].w);
                }
            }
        }
    }
}
int main()
{
    int x,y,w;
    while(scanf("%d%d%d",&n,&s,&m)!=EOF)
    {
        pos=0;
        memset(fa,-1,sizeof(fa));
        for(int i=0;i<n-1;i++)
        {
            scanf("%d%d%d",&x,&y,&w);
            e[pos].v=y;e[pos].w=w;e[pos].next=fa[x];fa[x]=pos++;
            e[pos].v=x;e[pos].w=w;e[pos].next=fa[y];fa[y]=pos++;
        }
        memset(dp,0,sizeof(dp));
        DP(s,-1);
        cout<<dp[s][m]<<'\n';
```

```
    }
    return 0;
}
```

7.2 DP 常用优化

DP 斜率优化

题意是给你一个 n 长度递增数列，将其分组，每组不少于 m 个，每组的 cost 是每组中所有元素减去里面最小元素的值的总和

得到方程：

dp[i]=min(dp[tem] -sum[tem]-(i-tem)*a[tem+1]) +sum[i];m<tem<i;

然后将 i 和 tem 分离

dp[i]=min(dp[tem] -sum[tem]+tem *a[tem+1]-i* a[tem+1])
+sum[i];m<tem<i;

可设 y= dp[tem] -sum[tem]+tem *a[tem+1](都是关于 tem 的)；

X= a[tem+1]; k=I;G= dp[tem] -sum[tem]+tem *a[tem+1]-i* a[tem+1];

然后我们可以知道 G=y-k*x;进而转化为 y=k*x+G;

显然 k 是大于 0 的，G 为该函数的 y 轴截距，要使截距最小。

这个式子中，G 是未知的，G 和 y 以及 x 有关。

如果我们把每个 j 对应的 x 和 y 值看成一个坐标系中的点的话。

那么当我们枚举到 i 时，坐标系中就有一系列的点。
对于每个点，做一条斜率为 k 的直线，就能得到一个 G 的值。G 的值为这条直线与 y 轴交点的纵坐标。
我们可以看到，实际上，如果我们让 G 的值由负无穷变化到正无穷，相当于一条直线，它满足斜率为 k，然后从坐标系的下方慢慢地向上平移到坐标系的上方。
那么，我们要找到，G 的最小值，就是在这个过程中，这条直线所碰到的第一个点！
而这个点，必然是这个点集的凸包上的点。

总结一下凸包维护规律：

斜率为负，从小到大，维护 U 左半部分凸包，求最小 y 截距

斜率为负，从大到小，维护倒 U 右半部分凸包，求最大 y 截距

斜率为正，从小到大，维护 U 右半部分凸包，求最小 y 截距

斜率为正，从大到小，维护倒 U 左半部分凸包，求最大 y 截距

注意斜率尽量中乘法，不要中实数，如有必要全用 long long，以免溢出。

注意 x2 - x1 可能小于 0 不等式的符号要改变（因为这个错了无数的题）

```
long long  dp[500005],sum[500005],a[500005];
long long G(long long x,long long y)
{
    return dp[y]+sum[x]+y*a[y+1]-sum[y]-dp[x]-x*a[x+1];
}
long long S(long long x,long long y)
{
    return a[y+1]-a[x+1];//这题给的是升序的所以一定大于=0，所以不用担心符号改
变
}
long long q[500005];
int r,f;
void insert(int k)
{
    q[++r]=k;
    for(long long i=r-1;i>f+1;i--)
    {
        long long x=q[i-1],y=q[i],z=q[i+1];
        if((G(x,y)*S(y,z)>=G(y,z)*S(x,y)))
            q[i]=q[r--];
        else
            break;
    }

}
void init()
{
    r=f=0;
    memset(q,0,sizeof(q));
    memset(dp,0,sizeof(dp));
}
int main()
{
```

```
    int c,n,m;
    scanf("%d",&c);
    while(c--)
    {
        scanf("%d%d",&n,&m);
        sum[0]=0;init();
        for(int i=1;i<=n;i++)
        {
            scanf("%lld",&a[i]);
            sum[i]=sum[i-1]+a[i];
        }
        for(long long i=1;i<=n;i++)
        {
            while(f+1<r&&G(q[f+1],q[f+2])<=i*S(q[f+1],q[f+2]))
                f++;
            long long tem=f<r?q[f+1]:0;
            dp[i]=dp[tem]+sum[i]-sum[tem]-(i-tem)*a[tem+1];
            if(i>=2*m-1)
            {
                insert(i-m+1);
            }
        }
        printf("%lld\n",dp[n]);
    }
    return 0;
}
```

四边形不等式

状态转移方程

d[i, j] = min{ d[i, k - 1] + d[k + 1][j] } + w[i , j]

i <= k <= j

时间复杂度为 O(n * n * n)。

如果函数 w 满足： w[a, c] + w[b, d] <= w[b, c] + w[a, d]

a < b < c < d

则说 w 满足凸四边形不等式（简称 w 为凸）。

如果函数 w 满足：w[i, j] <= w[i", j"]　　[i, j] 包含于 [i", j"]

则说 w 关于区间包含关系单调。

定理一：

如果 w 同时满足四边形不等式和区间单调关系，则 d 也满足四边形不等式；

定理二：

定理一的条件满足时让 d[i, j] 取最小值的 k 为 K[i, j]，则 K[i, j - 1] <= K[i, j] <= K[i + 1, j]；

定理三：

w 为凸当且仅当 w[i, j] + w[i + 1, j + 1] <= w[i + 1, j] + w[i, j + 1]。

这样每次决策范围变成 K[i + 1, j] 到 K[i, j - 1]。
按 j - i 递减的顺序递推各个状态值，则对于每个确定的 j - i 来说，决策总量为 O(n)，故总的时间复杂度为 O(n*n)。

转移方程 dp[i][j]=min(dp[i-1][k]+cost[k+1][j])(i-1<k<j);

```
const int maxn=1005;
long long cost[maxn][maxn],dp[maxn][maxn],s[maxn][maxn];
void DP(int n,int m)
{
    long long ans=1000000000000;
    for(int i=0;i<n;i++)
    {
        dp[0][i]=cost[0][i];
        s[0][i]=0;
        s[i][n]=n-2;
    }
    for(int i=1;i<=m;i++)
        for(int j=n-1;j>=0;j--)
        {
            long long tem=10000000000;
            int bin;
            for(int k=s[i-1][j];k<=s[i][j+1];k++)
            {
                if(tem>dp[i-1][k]+cost[k+1][j])
                {
                    tem=dp[i-1][k]+cost[k+1][j];
                    bin=k;
                }
            }
            s[i][j]=bin;
            dp[i][j]=tem;
        }
        printf("%lld\n",dp[m][n-1]);
```

```
}
int main()
{
    int n,m;
    long long sum[maxn],tem[maxn];
    while(scanf("%d%d",&n,&m)&&n&&m)
    {
        for(int i=0;i<n;i++)
        {
            scanf("%lld",&tem[i]);
            sum[i]=i==0?tem[i]:(sum[i-1]+tem[i]);
        }
        for(int i=1;i<n;i++)
            cost[0][i]=cost[0][i-1]+sum[i-1]*tem[i];
        for(int i=1;i<n;i++)
            for(int j=i+1;j<n;j++)
            {
                cost[i][j]=cost[i][j-1]+(sum[j-1]-sum[i-1])*tem[j];
            }
            DP(n,m);
    }
    return 0;
}
```

7.3 背包

```
0-1 背包
//m 为容量，n 为东西个数
memset(bag,0,sizeof(bag));
for(int i=1; i<=n; i++)
    for(int k=m; k>=w[i]; k--)
        if( bag[k-w[i]]+ v[i] > bag[k] )
            bag[k] = bag[k-w[i]]+ v[i];
cout << bag[m] << endl;


完全背包
for(int i=1; i<=n; i++)
    for(int k=w[i]; k<=c; k++)
```

```
        {
            money[k] = max( money[k], money[k-w[i]]+w[i] );
        }
cout << money[c] << endl;
```

多重背包

```
count = 1;
memset(hp,0,sizeof(hp));
for(int i=1; i<=n; i++)
    cin >> mp[i] >> hurt[i] >> num[i];
// mp 为每个东西体积，hurt 为价值，num 为数目
for(int i=1; i<=n; i++)
{
    int sum = 0;
    for(int k=0; ; k++)
    {
        int x = (int)pow(2,k);   // <math.h>
        if( sum + x > num[i] )
            break;
        sum += x;
        h[count] = x*hurt[i];
        mmp[count] = x*mp[i];
        count++;
    }
    int x = num[i] - sum;
    if( x == 0 )
        continue;
    h[count] = x*hurt[i];
    mmp[count] = x*mp[i];
    count++;
}
for(int i=1; i<count; i++)
    for(int k=m; k>=mmp[i]; k--)
        if( hp[k] < hp[k-mmp[i]] + h[i] )
            hp[k] = hp[k-mmp[i]] + h[i];
cout << hp[m] << endl;
```

P04: 混合三种背包问题

问题

如果将 **P01**、**P02**、**P03** 混合起来。也就是说，有的物品只可以取一次（**01** 背包），有

的物品可以取无限次（完全背包），有的物品可以取的次数有一个上限（多重背包）。应该怎么求解呢？

01 背包与完全背包的混合

考虑到在 **P01** 和 **P02** 中最后给出的伪代码只有一处不同，故如果只有两类物品：一类物品只能取一次，另一类物品可以取无限次，那么只需在对每个物品应用转移方程时，根据物品的类别选用顺序或逆序的循环即可，复杂度是 **O(VN)**。伪代码如下：

```
for i=1..N
if 第 i 件物品是 01 背包
for v=V..0
f[v]=max{f[v],f[v-c[i]]+w[i]};
else if 第 i 件物品是完全背包
for v=0..V
f[v]=max{f[v],f[v-c[i]]+w[i]};
```

再加上多重背包

如果再加上有的物品最多可以取有限次，那么原则上也可以给出 **O(VN)**的解法：遇到多重背包类型的物品用单调队列解即可。但如果不考虑超过 **NOIP** 范围的算法的话，用 **P03** 中将每个这类物品分成 **O(log n[i])**个 **01** 背包的物品的方法也已经很优了。

P05: 二维费用的背包问题

问题

二维费用的背包问题是指：对于每件物品，具有两种不同的费用；选择这件物品必须同时付出这两种代价；对于每种代价都有一个可付出的最大值（背包容量）。问怎样选择物品可以得到最大的价值。设这两种代价分别为代价 **1** 和代价 **2**，第 **i** 件物品所需的两种代价分别为 **a[i]**和 **b[i]**。两种代价可付出的最大值（两种背包容量）分别为 **V** 和 **U**。物品的价值为 **w[i]**。

算法

费用加了一维，只需状态也加一维即可。设 **f[i][v][u]**表示前 **i** 件物品付出两种代价分别为 **v** 和 **u** 时可获得的最大价值。状态转移方程就是：**f[i][v][u]=max{f[i-1][v][u],f[i-1][v-a[i]][u-b[i]]+w[i]}**。如前述方法，可以只使用二维的数组：当每件物品只可以取一次时变量 **v** 和 **u** 采用顺序的循环，当物品有如完全背包问题时采用逆序的循环。当物品有如多重背包问题时拆分物品。

物品总个数的限制

有时，“二维费用”的条件是以这样一种隐含的方式给出的：最多只能取 **M** 件物品。这事实上相当于每件物品多了一种“件数”的费用，每个物品的件数费用均为 **1**，可以付出的最大件数费用为 **M**。换句话说，设 **f[v][m]**表示付出费用 **v**、最多选 **m** 件时可得到的最大价值，则根据物品的类型（**01**、完全、多重）用不同的方法循环更新，最后在 **f[0..V][0..M]** 范围内寻找答案。

另外，如果要求“恰取 **M** 件物品”，则在 **f[0..V][M]**范围内寻找答案。

P06: 分组的背包问题

问题

有 **N** 件物品和一个容量为 **V** 的背包。第 **i** 件物品的费用是 **c[i]**，价值是 **w[i]**。这些物品被划分为若干组，每组中的物品互相冲突，最多选一件。求解将哪些物品装入背包可使这些物品的费用总和不超过背包容量，且价值总和最大。

算法

这个问题变成了每组物品有若干种策略：是选择本组的某一件，还是一件都不选。也就是说设 **f[k][v]**表示前 **k** 组物品花费费用 **v** 能取得的最大权值，则有 **f[k][v]=max{f[k-1][v],f[k-1][v-c[i]]+w[i]|**物品 **i** 属于第 **k** 组**}**。

使用一维数组的伪代码如下：

```
for 所有的组 k
for 所有的 i 属于组 k
for v=V..0
f[v]=max{f[v],f[v-c[i]]+w[i]}
```

另外，显然可以对每组中的物品应用 **P02** 中"一个简单有效的优化"。

P07: 有依赖的背包问题

简化的问题

这种背包问题的物品间存在某种"依赖"的关系。也就是说，**i** 依赖于 **j**，表示若选物品 **i**，则必须选物品 **j**。为了简化起见，我们先设没有某个物品既依赖于别的物品，又被别的物品所依赖；另外，没有某件物品同时依赖多件物品。

算法

这个问题由 **NOIP2006** 金明的预算方案一题扩展而来。遵从该题的提法，将不依赖于别的物品的物品称为"主件"，依赖于某主件的物品称为"附件"。由这个问题的简化条件可知所有的物品由若干主件和依赖于每个主件的一个附件集合组成。

按照背包问题的一般思路，仅考虑一个主件和它的附件集合。可是，可用的策略非常多，包括：一个也不选，仅选择主件，选择主件后再选择一个附件，选择主件后再选择两个附件**……**无法用状态转移方程来表示如此多的策略。（事实上，设有 **n** 个附件，则策略有 **2^n+1** 个，为指数级。）

考虑到所有这些策略都是互斥的（也就是说，你只能选择一种策略），所以一个主件和它的附件集合实际上对应于 **P06** 中的一个物品组，每个选择了主件又选择了若干个附件的策略对应于这个物品组中的一个物品，其费用和价值都是这个策略中的物品的值的和。但仅仅是这一步转化并不能给出一个好的算法，因为物品组中的物品还是像原问题的策略一样多。

再考虑 **P06** 中的一句话： 可以对每组中的物品应用 **P02** 中"一个简单有效的优化"。这提示我们，对于一个物品组中的物品，所有费用相同的物品只留一个价值最大的，不影响

结果。所以，我们可以对主件 **i** 的“附件集合”先进行一次 **01** 背包，得到费用依次为 **0..V-c[i]** 所有这些值时相应的最大价值 **f'[0..V-c[i]]**。那么这个主件及它的附件集合相当于 **V-c[i]+1** 个物品的物品组，其中费用为 **c[i]+k** 的物品的价值为 **f'[k]+w[i]**。也就是说原来指数级的策略中有很多策略都是冗余的，通过一次 **01** 背包后，将主件 **i** 转化为 **V-c[i]+1** 个物品的物品组，就可以直接应用 **P06** 的算法解决问题了。

更一般的问题

更一般的问题是：依赖关系以图论中“森林”的形式给出（森林即多叉树的集合），也就是说，主件的附件仍然可以具有自己的附件集合，限制只是每个物品最多只依赖于一个物品（只有一个主件）且不出现循环依赖。

解决这个问题仍然可以用将每个主件及其附件集合转化为物品组的方式。唯一不同的是，由于附件可能还有附件，就不能将每个附件都看作一个一般的 **01** 背包中的物品了。若这个附件也有附件集合，则它必定要被先转化为物品组，然后用分组的背包问题解出主件及其附件集合所对应的附件组中各个费用的附件所对应的价值。

事实上，这是一种树形 **DP**，其特点是每个父节点都需要对它的各个儿子的属性进行一次 **DP** 以求得自己的相关属性。这已经触及到了“泛化物品”的思想。看完 **P08** 后，你会发现这个“依赖关系树”每一个子树都等价于一件泛化物品，求某节点为根的子树对应的泛化物品相当于求其所有儿子的对应的泛化物品之和。

P08: 泛化物品

定义

考虑这样一种物品，它并没有固定的费用和价值，而是它的价值随着你分配给它的费用而变化。这就是泛化物品的概念。

更严格的定义之。在背包容量为 **V** 的背包问题中，泛化物品是一个定义域为 **0..V** 中的整数的函数 **h**，当分配给它的费用为 **v** 时，能得到的价值就是 **h(v)**。

这个定义有一点点抽象，另一种理解是一个泛化物品就是一个数组 **h[0..V]**，给它费用 **v**，可得到价值 **h[V]**。

一个费用为 **c** 价值为 **w** 的物品，如果它是 **01** 背包中的物品，那么把它看成泛化物品，它就是除了 **h(c)=w** 其它函数值都为 **0** 的一个函数。如果它是完全背包中的物品，那么它可以看成这样一个函数，仅当 **v** 被 **c** 整除时有 **h(v)=v/c*w**，其它函数值均为 **0**。如果它是多重背包中重复次数最多为 **n** 的物品，那么它对应的泛化物品的函数有 **h(v)=v/c*w** 仅当 **v** 被 **c** 整除且 **v/c<=n**，其它情况函数值均为 **0**。

一个物品组可以看作一个泛化物品 **h**。对于一个 **0..V** 中的 **v**，若物品组中不存在费用为 **v** 的的物品，则 **h(v)=0**，否则 **h(v)**为所有费用为 **v** 的物品的最大价值。**P07** 中每个主件及其附件集合等价于一个物品组，自然也可看作一个泛化物品。

泛化物品的和

如果面对两个泛化物品 **h** 和 **l**，要用给定的费用从这两个泛化物品中得到最大的价值，怎么求呢？事实上，对于一个给定的费用 **v**，只需枚举将这个费用如何分配给两个泛化物品就可以了。同样的，对于 **0..V** 的每一个整数 **v**，可以求得费用 **v** 分配到 **h** 和 **l** 中的最大价值 **f(v)**。也即 **f(v)=max{h(k) +l(v-k)|0<=k<=v}**。可以看到，**f** 也是一个由泛化物品 **h** 和 **l** 决定的定义域为 **0..V** 的函数，也就是说，**f** 是一个由泛化物品 **h** 和 **l** 决定的泛化物品。

由此可以定义泛化物品的和：**h**、**l** 都是泛化物品，若泛化物品 **f** 满足 **f(v)=max{h(k)+l(v-k)|0<=k<=v}**，则称 **f** 是 **h** 与 **l** 的和，即 **f=h+l**。这个运算的时间复杂度是 **O(V^2)**。

泛化物品的定义表明：在一个背包问题中，若将两个泛化物品代以它们的和，不影响问题的答案。事实上，对于其中的物品都是泛化物品的背包问题，求它的答案的过程也就是求所有这些泛化物品之和的过程。设此和为 **s**，则答案就是 **s[0..V]**中的最大值。

背包问题的泛化物品

一个背包问题中，可能会给出很多条件，包括每种物品的费用、价值等属性，物品之间的分组、依赖等关系等。但肯定能将问题对应于某个泛化物品。也就是说，给定了所有条件以后，就可以对每个非负整数 **v** 求得：若背包容量为 **v**，将物品装入背包可得到的最大价值是多少，这可以认为是定义在非负整数集上的一件泛化物品。这个泛化物品——或者说问题所对应的一个定义域为非负整数的函数——包含了关于问题本身的高度浓缩的信息。一般而言，求得这个泛化物品的一个子域（例如 **0..V**）的值之后，就可以根据这个函数的取值得到背包问题的最终答案。

综上所述，一般而言，求解背包问题，即求解这个问题所对应的一个函数，即该问题的泛化物品。而求解某个泛化物品的一种方法就是将它表示为若干泛化物品的和然后求之。