一、数学问题 4
1.精度计算—大数阶乘 4
6.任意进制转换 4
7.最大公约数、最小公倍数 5
8.组合序列 6
9.快速傅立叶变换（FFT） 6
10.Ronberg 算法计算积分 8
11.行列式计算 10
12.求排列组合数 10
13.求某一天星期几 11
14.卡特兰（Catalan）数列 原理 11
二、字符串处理 12
1.字符串替换 12
2.字符串查找 13
6.数字转换为字符 13
三、计算几何 14
1.叉乘法求任意多边形面积 14
2.求三角形面积 14
3.两矢量间角度 15
4.两点距离（2D、3D） 15
5.射向法判断点是否在多边形内部 16
6.判断点是否在线段上 17
7.判断两线段是否相交 18
8.判断线段与直线是否相交 18
9.点到线段最短距离 19
10.求两直线的交点 20
11.判断一个封闭图形是凹集还是凸集 20
12.Graham 扫描法寻找凸包 21
13.求两条线段的交点 23
四、数论 23
1.x 的二进制长度 23

2.返回 x 的二进制表示中从低到高的第 i 位 24
3.模取幂运算 24
4.求解模线性方程 24
5.求解模线性方程组(中国余数定理) 25
8.求距阵最大和 26
10.质因数分解 27
11.高斯消元法解线性方程组 28
五、图论 28
5.解欧拉图 28
八、高精度运算专题 30
1.专题函数说明 30
2.高精度数比较 30
5.高精度乘 10 31
8.高精度除单精度 31
9.高精度除高精度 32
my own 33
单调队列 33
KMP 算法 35
多关键字排序 37
次短路（SPFA） 37
快速幂 40
Huffman 41
求 n 以内质数 42
求组合数 42
二分答案模版 45
最优子矩阵 45
连续最大和 47
RMQ 48
ACM 会用到的一点数学知识 49
计数排序 50
高精度加法 50

高精度减法 51
高精度乘法 53
归并排序 54
位运算 55
逆矩阵 56
行列式 58
floyd 求最小环 59
other 59

一、数学问题

1.精度计算—大数阶乘

语法：int result=factorial(int n);
参数：
n：n 的阶乘
返回值：阶乘结果的位数
注意：
本程序直接输出 n!的结果，需要返回结果请保留 long a[] 需要 math.h
源程序：

```
int factorial(int n)
{
long a[10000];
int i,j,l,c,m=0,w;
a[0]=1;
for(i=1;i<=n;i++)
    {
    c=0;
    for(j=0;j<=m;j++)
        {
        a[j]=a[j]*i+c;
        c=a[j]/10000;
        a[j]=a[j]%10000;
        }
    if(c>0) {m++;a[m]=c;}
   }
w=m*4+log10(a[m])+1;
printf("\n%ld",a[m]);
for(i=m-1;i>=0;i--) printf("%4.4ld",a[i]);
return w;
}
```

6.任意进制转换

语法：conversion(char s1[],char s2[],char t[]);
参数：
s[]：转换前的数字
s2[]：转换后的数字
d1：原进制数
d2：需要转换到的进制数
返回值：null
注意：

高于 9 的位数用大写'A'~'Z'表示，2~16 位进制通过验证

源程序：

```
 void conversion(char s[],char s2[],long d1,long d2)
{
    long i,j,t,num;
    char c;
    num=0;
    for (i=0;s[i]!='\0';i++)
        {
        if (s[i]<='9'&&s[i]>='0') t=s[i]-'0'; else t=s[i]-'A'+10;
        num=num*d1+t;
        }
    i=0;
    while(1)
        {
        t=num%d2;
        if (t<=9) s2[i]=t+'0'; else s2[i]=t+'A'-10;
        num/=d2;
        if (num==0) break;
        i++;
        }
    for (j=0;j<i/2;j++)
        {c=s2[j];s2[j]=s[i-j];s2[i-j]=c;}
    s2[i+1]='\0';
}
```

7.最大公约数、最小公倍数

语法：resulet=hcf(int a,int b)、result=lcd(int a,int b)

参数：

a：int a，求最大公约数或最小公倍数

b：int b，求最大公约数或最小公倍数

返回值：返回最大公约数（hcf）或最小公倍数（lcd）

注意： lcd 需要连同 hcf 使用

源程序：

```
 int hcf(int a,int b)
{
    int r=0;
    while(b!=0)
        {
        r=a%b;
        a=b;
        b=r;
        }
    return(a);
```

```
}
lcd(int u,int v,int h)
{
    return(u*v/h);
}
```

8.组合序列

语法：m_of_n(int m, int n1, int m1, int* a, int head)

参数：

m：组合数 C 的上参数

n1：组合数 C 的下参数

m1：组合数 C 的上参数，递归之用

*a：1～n 的整数序列数组

head：头指针

返回值：null

注意：

*a 需要自行产生

初始调用时，m=m1、head=0

调用例子：求 C(m,n)序列：m_of_n(m,n,m,a,0);

源程序：

```
void m_of_n(int m, int n1, int m1, int* a, int head)
{
    int i,t;
    if(m1<0 || m1>n1) return;
    if(m1==n1)
        {
        return;
        }
    m_of_n(m,n1-1,m1,a,head); // 递归调用
    t=a[head];a[head]=a[n1-1+head];a[n1-1+head]=t;
    m_of_n(m,n1-1,m1-1,a,head+1); // 再次递归调用
    t=a[head];a[head]=a[n1-1+head];a[n1-1+head]=t;
}
```

9.快速傅立叶变换（FFT）

语法：kkfft(double pr[],double pi[],int n,int k,double fr[],double fi[],int l,int il);

参数：

pr[n]：输入的实部

pi[n]：数入的虚部

n，k：满足 n=2^k

fr[n]：输出的实部

fi[n]：输出的虚部

l：逻辑开关，0 FFT，1 ifFT

il：逻辑开关，0 输出按实部/虚部；1 输出按模/幅角

返回值：null

注意： 需要 math.h

源程序：

```
 void kkfft(pr,pi,n,k,fr,fi,l,il)
int n,k,l,il;
double pr[],pi[],fr[],fi[];
{
    int it,m,is,i,j,nv,l0;
    double p,q,s,vr,vi,poddr,poddi;
    for (it=0; it<=n-1; it++)
        {
         m=it; is=0;
        for (i=0; i<=k-1; i++)
            {j=m/2; is=2*is+(m-2*j); m=j;}
        fr[it]=pr[is]; fi[it]=pi[is];
        }
    pr[0]=1.0; pi[0]=0.0;
    p=6.283185306/(1.0*n);
    pr[1]=cos(p); pi[1]=-sin(p);
    if (l!=0) pi[1]=-pi[1];
    for (i=2; i<=n-1; i++)
        {
       p=pr[i-1]*pr[1];
       q=pi[i-1]*pi[1];
        s=(pr[i-1]+pi[i-1])*(pr[1]+pi[1]);
        pr[i]=p-q; pi[i]=s-p-q;
        }
    for (it=0; it<=n-2; it=it+2)
        {
       vr=fr[it]; vi=fi[it];
        fr[it]=vr+fr[it+1]; fi[it]=vi+fi[it+1];
        fr[it+1]=vr-fr[it+1]; fi[it+1]=vi-fi[it+1];
        }
    m=n/2; nv=2;
    for (l0=k-2; l0>=0; l0--)
        {
        m=m/2; nv=2*nv;
        for (it=0; it<=(m-1)*nv; it=it+nv)
            for (j=0; j<=(nv/2)-1; j++)
                {
               p=pr[m*j]*fr[it+j+nv/2];
                q=pi[m*j]*fi[it+j+nv/2];
                s=pr[m*j]+pi[m*j];
```

```
                s=s*(fr[it+j+nv/2]+fi[it+j+nv/2]);
                poddr=p-q; poddi=s-p-q;
                fr[it+j+nv/2]=fr[it+j]-poddr;
                fi[it+j+nv/2]=fi[it+j]-poddi;
                fr[it+j]=fr[it+j]+poddr;
                fi[it+j]=fi[it+j]+poddi;
                }
        }
    if (l!=0)
        for (i=0; i<=n-1; i++)
            {
           fr[i]=fr[i]/(1.0*n);
            fi[i]=fi[i]/(1.0*n);
            }
    if (il!=0)
            for (i=0; i<=n-1; i++)
            {
           pr[i]=sqrt(fr[i]*fr[i]+fi[i]*fi[i]);
            if (fabs(fr[i])<0.000001*fabs(fi[i]))
                {
               if ((fi[i]*fr[i])>0) pi[i]=90.0;
                else pi[i]=-90.0;
                }
            else
                pi[i]=atan(fi[i]/fr[i])*360.0/6.283185306;
            }
    return;
}
```

10.Ronberg 算法计算积分

语法：result=integral(double a,double b);

参数：

a：积分上限

b：积分下限

function f：积分函数

返回值：f 在 (a,b) 之间的积分值

注意：

function f(x)需要自行修改，程序中用的是 sina(x)/x

需要 math.h

默认精度要求是 1e-5

源程序：

```
double f(double x)
{
    return sin(x)/x; //在这里插入被积函数
```

```
}

double integral(double a,double b)
{
    double h=b-a;
    double t1=(1+f(b))*h/2.0;
    int k=1;
    double r1,r2,s1,s2,c1,c2,t2;
loop:
    double s=0.0;
    double x=a+h/2.0;
    while(x<b)
        {
        s+=f(x);
        x+=h;
        }
    t2=(t1+h*s)/2.0;
    s2=t2+(t2-t1)/3.0;
    if(k==1)
      {
        k++;h/=2.0;t1=t2;s1=s2;
        goto loop;
        }
    c2=s2+(s2-s1)/15.0;
    if(k==2){
        c1=c2;k++;h/=2.0;
        t1=t2;s1=s2;
        goto loop;
        }
    r2=c2+(c2-c1)/63.0;
    if(k==3){
        r1=r2; c1=c2;k++;
        h/=2.0;
        t1=t2;s1=s2;
        goto loop;
        }
    while(fabs(1-r1/r2)>1e-5){
        r1=r2;c1=c2;k++;
        h/=2.0;
        t1=t2;s1=s2;
        goto loop;
        }
    return r2;
}
```

11.行列式计算

语法：result=js(int s[][],int n)

参数：

s[][]：行列式存储数组

n：行列式维数，递归用

返回值：行列式值

注意： 函数中常数 N 为行列式维度，需自行定义

源程序：

```
 int js(s,n)
int s[][N],n;
{
    int z,j,k,r,total=0;
    int b[N][N];/*b[N][N]用于存放，在矩阵 s[N][N]中元素 s[0]的余子式*/
    if(n>2)
        {
        for(z=0;z<n;z++)
            {
            for(j=0;j<n-1;j++)
                 for(k=0;k<n-1;k++)
                       if(k>=z) b[j][k]=s[j+1][k+1];  else
b[j][k]=s[j+1][k];
            if(z%2==0) r=s[0][z]*js(b,n-1); /*递归调用*/
            else r=(-1)*s[0][z]*js(b,n-1);
            total=total+r;
            }
        }
    else if(n==2)
       total=s[0][0]*s[1][1]-s[0][1]*s[1][0];
    return total;
}
```

12.求排列组合数

语法：result=P(long n,long m); / result=long C(long n,long m);

参数：

m：排列组合的上系数

n：排列组合的下系数

返回值：排列组合数

注意： 符合数学规则：m< = n

源程序：

```
 long P(long n,long m)
{
    long p=1;
    while(m!=0)
```

```
        {p*=n;n--;m--;}
    return p;
}
long C(long n,long m)
{
    long i,c=1;
    i=m;
    while(i!=0)
        {c*=n;n--;i--;}
    while(m!=0)
        {c/=m;m--;}
    return c;
}
```

13.求某一天星期几

语法：result=weekday(int N,int M,int d)

参数：

N,M,d：年月日，例如：2003,11,4

返回值：0：星期天，1 星期一……

注意： 需要 math.h ， 适用于 1582 年 10 月 15 日之后，因为罗马教皇格里高利十三世在这一天启用新历法.

源程序：

```
 int weekday(int N,int M,int d)
{
int m,n,c,y,w;
m=(M-2)%12;
if (M>=3) n=N;else n=N-1;
c=n/100;
y=n%100;
w=(int)(d+floor(13*m/5)+y+floor(y/4)+floor(c/4)-2*c)%7;
while(w<0) w+=7;
return w;
}
```

14.卡特兰（Catalan）数列 原理

令 h(1)＝1，catalan 数满足递归式：

h(n)= h(1)*h(n-1) + h(2)*h(n-2) + ... + h(n-1)h(1) (其中 n>=2)

该递推关系的解为：h(n)=c(2n-2,n-1)/n (n=1,2,3,...)

1, 1, 2, 5, 14, 42, 132, 429, 1430, 4862, 16796, 58786, 208012, 742900, 2674440, 9694845, 35357670, 129644790, 477638700, 1767263190, 6564120420, 24466267020, 91482563640, 343059613650, 1289904147324, 4861946401452, …

1.括号化问题。

矩阵链乘： P=a1×a2×a3×……×an，依据乘法结合律，不改变其顺序，只用括号表示成对的乘积，试问有几种括号化的方案？(h(n)种)

2.出栈次序问题。

一个栈(无穷大)的进栈序列为 1,2,3,..n,有多少个不同的出栈序列?

类似：有 2n 个人排成一行进入剧场。入场费 5 元。其中只有 n 个人有一张 5 元钞票，另外 n 人只有 10 元钞票，剧院无其它钞票，问有多少中方法使得只要有 10 元的人买票，售票处就有 5 元的钞票找零？(将持 5 元者到达视作将 5 元入栈，持 10 元者到达视作使栈中某 5 元出栈)

3.将多边行划分为三角形问题。

将一个凸多边形区域分成三角形区域的方法数?

类似：一位大城市的律师在她住所以北 n 个街区和以东 n 个街区处工作。每天她走 2n 个街区去上班。如果他从不穿越（但可以碰到）从家到办公室的对角线，那么有多少条可能的道路？

类似：在圆上选择 2n 个点,将这些点成对连接起来使得所得到的 n 条线段不相交的方法数?

二、字符串处理

1.字符串替换

语法：replace(char str[],char key[],char swap[]);

参数：

str[]：在此源字符串进行替换操作

key[]：被替换的字符串，不能为空串

swap[]：替换的字符串，可以为空串，为空串表示在源字符中删除 key[]

返回值：null

注意：

默认 str[]长度小于 1000，如否，重新设定设定 tmp 大小

需要 string.h

源程序：

```
 void replace(char str[],char key[],char swap[])
{
    int l1,l2,l3,i,j,flag;
    char tmp[1000];
    l1=strlen(str);
    l2=strlen(key);
    l3=strlen(swap);
    for (i=0;i<=l1-l2;i++)
        {
        flag=1;
        for (j=0;j<l2;j++)
            if (str[i+j]!=key[j]) {flag=0;break;}
        if (flag)
            {
            strcpy(tmp,str);
            strcpy(&tmp[i],swap);
            strcpy(&tmp[i+l3],&str[i+l2]);
            strcpy(str,tmp);
            i+=l3-1;
```

```
            l1=strlen(str);
            }
        }
}
```

2.字符串查找

语法：result=strfind(char str[],char key[]);

参数：

str[]：在此源字符串进行查找操作

key[]：被查找的字符串，不能为空串

返回值：如果查找成功，返回 key 在 str 中第一次出现的位置，否则返回-1

注意： 需要 string.h

源程序：

```
 int strfind(char str[],char key[])
{
    int l1,l2,i,j,flag;
    l1=strlen(str);
    l2=strlen(key);
    for (i=0;i<=l1-l2;i++)
        {
        flag=1;
        for (j=0;j<l2;j++)
            if (str[i+j]!=key[j]) {flag=0;break;}
        if (flag) return i;
        }
    return -1;
}
```

6.数字转换为字符

语法：cstr(int k,char o[]);

参数：

k：转换的数字

o[]：存储转换结果的字符串

返回值：null

注意： 需要 math.h

源程序：

```
 void cstr(int k,char o[])
{
    int len,i,t;
    len=log10(k)+1;
    for (i=len;i>0;i--)
        {
        t=k%10;
```

```
        k-=t;k/=10;
        o[i-1]='0'+t;
        }
    o[len]='\0';
}
```

三、计算几何

1.叉乘法求任意多边形面积

语法：result=polygonarea(Point *polygon,int N);

参数：

*polygon：多变形顶点数组

N：多边形顶点数目

返回值：多边形面积

注意：

支持任意多边形，凹、凸皆可

多边形顶点输入时按顺时针顺序排列

源程序：

```
typedef struct {
    double x,y;
} Point;
double polygonarea(Point *polygon,int N)
{
    int i,j;
    double area = 0;
    for (i=0;i<N;i++) {
        j = (i + 1) % N;
        area += polygon[i].x * polygon[j].y;
        area -= polygon[i].y * polygon[j].x;
        }
    area /= 2;
    return(area < 0 ? -area : area);
}
```

2.求三角形面积

语法：result=area3(float x1,float y1,float x2,float y2,float x3,float y3);

参数：

x1～3：三角形 3 个顶点 x 坐标

y1～3：三角形 3 个顶点 y 坐标

返回值：三角形面积

注意：

需要 math.h

源程序：

```
float area3(float x1,float y1,float x2,float y2,float x3,float y3)
{
    float a,b,c,p,s;
    a=sqrt((x1-x2)*(x1-x2)+(y1-y2)*(y1-y2));
    b=sqrt((x1-x3)*(x1-x3)+(y1-y3)*(y1-y3));
    c=sqrt((x3-x2)*(x3-x2)+(y3-y2)*(y3-y2));
    p=(a+b+c)/2;
    s=sqrt(p*(p-a)*(p-b)*(p-c));
    return s;
}
```

3.两矢量间角度

语法：result=angle(double x1, double y1, double x2, double y2);

参数：

x/y1～2：两矢量的坐标

返回值：两的角度矢量

注意：

返回角度为弧度制，并且以逆时针方向为正方向

需要 math.h

源程序：

```
#define PI 3.1415926

double angle(double x1, double y1, double x2, double y2)
{
    double dtheta,theta1,theta2;
    theta1 = atan2(y1,x1);
    theta2 = atan2(y2,x2);
    dtheta = theta2 - theta1;
    while (dtheta > PI)
        dtheta -= PI*2;
    while (dtheta < -PI)
        dtheta += PI*2;
    return(dtheta);
}
```

4.两点距离（2D、3D）

语法：result=distance_2d(float x1,float x2,float y1,float y2);

参数：

x/y/z1～2：各点的 x、y、z 坐标

返回值：两点之间的距离

注意：

需要 math.h

源程序：

```
float distance_2d(float x1,float x2,float y1,float y2)
{
    return(sqrt((x1-x2)*(x1-x2)+(y1-y2)*(y1-y2)));
}
float distance_3d(float x1,float x2,float y1,float y2,float z1,float z2)
{
    return(sqrt((x1-x2)*(x1-x2)+(y1-y2)*(y1-y2)+(z1-z2)*(z1-z2)));
}
```

5.射向法判断点是否在多边形内部

语法：result=insidepolygon(Point *polygon,int N,Point p);

参数：

*polygon：多边形顶点数组

N：多边形顶点个数

p：被判断点

返回值：0：点在多边形内部；1：点在多边形外部

注意：

若 p 点在多边形顶点或者边上，返回值不确定，需另行判断

需要 math.h

源程序：

```
#define MIN(x,y) (x < y ? x : y)
#define MAX(x,y) (x > y ? x : y)
typedef struct {
    double x,y;
} Point;
int insidepolygon(Point *polygon,int N,Point p)
{
    int counter = 0;
    int i;
    double xinters;
    Point p1,p2;
    p1 = polygon[0];
    for (i=1;i<=N;i++) {
        p2 = polygon[i % N];
        if (p.y > MIN(p1.y,p2.y)) {
            if (p.y <= MAX(p1.y,p2.y)) {
                if (p.x <= MAX(p1.x,p2.x)) {
                    if (p1.y != p2.y) {
                        xinters = (p.y-p1.y)*(p2.x-p1.x)/(p2.y-p1.y)+p1.x;
                        if (p1.x == p2.x || p.x <= xinters)
                            counter++;
                        }
```

```
                }
            }
        }
        p1 = p2;
    }
    if (counter % 2 == 0)
        return(OUTSIDE);
    else
        return(INSIDE);
}
```

6.判断点是否在线段上

语法：result=Pointonline(Point p1,Point p2,Point p);

参数：

p1、p2：线段的两个端点

p：被判断点

返回值：0：点在不在线段上；1：点在线段上

注意：

若 p 线段端点上返回 1

需要 math.h

源程序：

```
#define MIN(x,y) (x < y ? x : y)
#define MAX(x,y) (x > y ? x : y)
typedef struct {
double x,y;
} Point;
int FC(double x1,double x2)
{
    if (x1-x2<0.000002&&x1-x2>-0.000002) return 1; else return 0;
}

int Pointonline(Point p1,Point p2,Point p)
{
    double x1,y1,x2,y2;
    x1=p.x-p1.x;
    x2=p2.x-p1.x;
    y1=p.y-p1.y;
    y2=p2.y-p1.y;
    if (FC(x1*y2-x2*y1,0)==0) return 0;
    if ((MIN(p1.x,p2.x)<=p.x&&p.x<=MAX(p1.x,p2.x))&&
            (MIN(p1.y,p2.y)<=p.y&&p.y<=MAX(p1.y,p2.y)))
        return 1; else return 0;
}
```

7.判断两线段是否相交

语法：result=lineintersect(Point p1,Point p2,Point p3,Point p4);

参数：

p1～4：两条线段的四个端点

返回值：0：两线段不相交；1：两线段相交；2 两线段首尾相接

注意：

p1!=p2;p3!=p4;

源程序：

```
#define MIN(x,y) (x < y ? x : y)
#define MAX(x,y) (x > y ? x : y)
typedef struct {
    double x,y;
} Point;
int lineintersect(Point p1,Point p2,Point p3,Point p4)
{
    Point tp1,tp2,tp3;
    if

((p1.x==p3.x&&p1.y==p3.y)||(p1.x==p4.x&&p1.y==p4.y)||(p2.x==p3.x&&p2.y==p3.y)||(p2.x==p4.x&&p2.y==p4.y))
        return 2;
//快速排斥试验
    if
((MIN(p1.x,p2.x)<=p3.x&&p3.x<=MAX(p1.x,p2.x)&&MIN(p1.y,p2.y)<=p3.y&&p3.y<=MAX(p1.y,p2.y))||

(MIN(p1.x,p2.x)<=p4.x&&p4.x<=MAX(p1.x,p2.x)&&MIN(p1.y,p2.y)<=p4.y&&p4.y<=MAX(p1.y,p2.y)))
       ;else return 0;
//跨立试验
    tp1.x=p1.x-p3.x;
    tp1.y=p1.y-p3.y;
    tp2.x=p4.x-p3.x;
    tp2.y=p4.y-p3.y;
    tp3.x=p2.x-p3.x;
    tp3.y=p2.y-p3.y;
    if ((tp1.x*tp2.y-tp1.y*tp2.x)*(tp2.x*tp3.y-tp2.y*tp3.x)>=0) return 1;
else return 0;
}
```

8.判断线段与直线是否相交

语法：result=lineintersect(Point p1,Point p2,Point p3,Point p4);

参数：

p1、p2：线段的两个端点

p3、p4：直线上的两个点

返回值：0：线段直线不相交；1：线段和直线相交

注意：

如线段在直线上，返回 1

源程序：

```
typedef struct {
   double x,y;
} Point;
int lineintersect(Point p1,Point p2,Point p3,Point p4)
{
    Point tp1,tp2,tp3;
    tp1.x=p1.x-p3.x;
    tp1.y=p1.y-p3.y;
    tp2.x=p4.x-p3.x;
    tp2.y=p4.y-p3.y;
    tp3.x=p2.x-p3.x;
    tp3.y=p2.y-p3.y;
    if ((tp1.x*tp2.y-tp1.y*tp2.x)*(tp2.x*tp3.y-tp2.y*tp3.x)>=0) return 1;
else return 0;
}
```

9.点到线段最短距离

语法：result=mindistance(Point p1,Point p2,Point q);

参数：

p1、p2：线段的两个端点

q：判断点

返回值：点 q 到线段 p1p2 的距离

注意：

需要 math.h

源程序：

```
#define MIN(x,y) (x < y ? x : y)
#define MAX(x,y) (x > y ? x : y)
typedef struct {
    double x,y;
} Point;
double mindistance(Point p1,Point p2,Point q)
{
    int flag=1;
    double k;
    Point s;
    if (p1.x==p2.x) {s.x=p1.x;s.y=q.y;flag=0;}
    if (p1.y==p2.y) {s.x=q.x;s.y=p1.y;flag=0;}
    if (flag)
        {
        k=(p2.y-p1.y)/(p2.x-p1.x);
        s.x=(k*k*p1.x+k*(q.y-p1.y)+q.x)/(k*k+1);
```

```
            s.y=k*(s.x-p1.x)+p1.y;
            }
        if (MIN(p1.x,p2.x)<=s.x&&s.x<=MAX(p1.x,p2.x))
            return sqrt((q.x-s.x)*(q.x-s.x)+(q.y-s.y)*(q.y-s.y));
        else
            return

MIN(sqrt((q.x-p1.x)*(q.x-p1.x)+(q.y-p1.y)*(q.y-p1.y)),sqrt((q.x-p2.x)*(q.x-p2.x)+(q.y-p2.y)*(q.y-p2.y)));
    }
```

10.求两直线的交点

语法：result=mindistance(Point p1,Point p2,Point q);

参数：

p1～p4：直线上不相同的两点

*p：通过指针返回结果

返回值：1：两直线相交；2：两直线平行

注意：

如需要判断两线段交点，检验 k 和对应 k1（注释中）的值是否在 0～1 之间，用在 0～1 之间的那个求交点

源程序：

```
 typedef struct {
   double x,y;
} Point;
int linecorss(Point p1,Point p2,Point p3,Point p4,Point *p)
{
   double k;
   if ((p4.y-p3.y)*(p2.x-p1.x)-(p4.x-p3.x)*(p2.y-p1.y)==0) return 0;
  if ((p4.x-p3.x)*(p1.y-p3.y)-(p4.y-p3.y)*(p1.x-p3.x)==0&&
          (p2.x-p1.x)*(p1.y-p3.y)-(p2.y-p1.y)*(p1.x-p3.x)==0) return 0;


k=((p4.x-p3.x)*(p1.y-p3.y)-(p4.y-p3.y)*(p1.x-p3.x))/((p4.y-p3.y)*(p2.x-p1.x)-(p4.x-p3.x)*(p2.y-p1.y));

//k1=((p2.x-p1.x)*(p1.y-p3.y)-(p2.y-p1.y)*(p1.x-p3.x))/((p4.y-p3.y)*(p2.x-p1.x)-(p4.x-p3.x)*(p2.y-p1.y));
   (*p).x=p1.x+k*(p2.x-p1.x);
   (*p).y=p1.y+k*(p2.y-p1.y);
   return 1;
}
```

11.判断一个封闭图形是凹集还是凸集

语法：result=convex(Point *p,int n);

参数：

*p：封闭曲线顶点数组

n：封闭曲线顶点个数

返回值：1：凸集；-1：凹集；0：曲线不符合要求无法计算

注意：

默认曲线为简单曲线：无交叉、无圈

源程序：

```
typedef struct {
   double x,y;
} Point;
int convex(Point *p,int n)
{
   int i,j,k;
   int flag = 0;
   double z;
   if (n < 3)
      return(0);
   for (i=0;i<n;i++) {
      j = (i + 1) % n;
      k = (i + 2) % n;
      z = (p[j].x - p[i].x) * (p[k].y - p[j].y);
      z -= (p[j].y - p[i].y) * (p[k].x - p[j].x);
      if (z < 0)
         flag |= 1;
      else if (z > 0)
         flag |= 2;
      if (flag == 3)
         return  - 1; //CONCAVE
      }
   if (flag != 0)
      return 1; //CONVEX
   else
   return 0;
}
```

12.Graham 扫描法寻找凸包

语法：Graham_scan(Point PointSet[],Point ch[],int n,int &len);

参数：

PointSet[]：输入的点集

ch[]：输出的凸包上的点集，按照逆时针方向排列

n：PointSet 中的点的数目

len：输出的凸包上的点的个数

返回值：null

源程序：

```
struct Point{
   float x,y;
};
```

```
float multiply(Point p1,Point p2,Point p0)
{
    return((p1.x-p0.x)*(p2.y-p0.y)-(p2.x-p0.x)*(p1.y-p0.y));
}
float distance(Point p1,Point p2)
{
    return(sqrt((p1.x-p2.x)*(p1.x-p2.x)+(p1.y-p2.y)*(p1.y-p2.y)));
}
void Graham_scan(Point PointSet[],Point ch[],int n,int &len)
{
    int i,j,k=0,top=2;
    Point tmp;

   for(i=1;i<n;i++)
    if
((PointSet[i].y<PointSet[k].y)||((PointSet[i].y==PointSet[k].y)&&(PointSet[i].x<PointSet[k].x)))
    k=i;
    tmp=PointSet[0];
    PointSet[0]=PointSet[k];
    PointSet[k]=tmp;
    for (i=1;i<n-1;i++)
        {
        k=i;
        for (j=i+1;j<n;j++)
            if ( (multiply(PointSet[j],PointSet[k],PointSet[0])>0) ||
                    ((multiply(PointSet[j],PointSet[k],PointSet[0])==0)

&&(distance(PointSet[0],PointSet[j])<distance(PointSet[0],PointSet[k])))
)
                k=j;
        tmp=PointSet[i];
        PointSet[i]=PointSet[k];
        PointSet[k]=tmp;
        }
    ch[0]=PointSet[0];
    ch[1]=PointSet[1];
    ch[2]=PointSet[2];
    for (i=3;i<n;i++)
        {
        while (multiply(PointSet[i],ch[top],ch[top-1])>=0) top--;
        ch[++top]=PointSet[i];
        }
    len=top+1;
}
```

13.求两条线段的交点

语法：Result=IntersectPoint (Point p1,Point p2,Point p3,Point p4,Point &p);

参数：

P1～P4：两条线断 4 个端点

P：线段交点

返回值：如果两条线段平行无交点，返回 0，否则返回 1

源程序：

```
 struct Point{
    float x,y;
};

int IntersectPoint (Point p1,Point p2,Point p3,Point p4,Point &p)
{
float a,b,c,d,e,f;
a=p2.y-p1.y;
b=p1.x-p2.x;
c=p1.y*(p2.x-p1.x)+p1.x*(p2.y-p1.y);
d=p4.y-p3.y;
e=p3.x-p4.x;
f=p3.y*(p4.x-p3.x)+p1.x*(p4.y-p3.y);
if (a*e==b*d)
    return 0;
else
    {
    p.x=(e*c-b*f)/(b*d-a*e);
    p.y=(d*c-a*f)/(a*e-b*d);
    return 1;
    }
}
```

四、数论

1.x 的二进制长度

语法：result=BitLength(int x);

参数：

x：测长的 x

返回值：x 的二进制长度

源程序：

```
 int BitLength(int x)
{
    int d = 0;
```

```
    while (x > 0) {
        x >>= 1;
        d++;
    }
    return d;
}
```

2.返回 x 的二进制表示中从低到高的第 i 位

语法：result=BitAt(int x, int i);

参数：

x：十进制 x

i：要求二进制的第 i 位

返回值：返回 x 的二进制表示中从低到高的第 i 位

注意：

最低位为第一位

源程序：

```
int BitAt(int x, int i)
{
    return ( x & (1 << (i-1)) );
}
```

3.模取幂运算

语法：result=Modular_Expoent(int a,int b,int n);

参数：

a、b、n：a^b mod n 的对应参数

返回值：a^b mod n 的值

注意：

需要 BitLength 和 BitAt

源程序：

```
int Modular_Expoent(int a,int b,int n)
{
    int i, y=1;
    for (i = BitLength(b); i > 0; i--)
        {
        y = (y*y)%n;
        if (BitAt(b,i) > 0)
        y = (y*a)%n;
        }
    return y;
}
```

4.求解模线性方程

语法：result = modular_equation(int a,int b,int n);

参数：

a、b、n：ax=b (mod n) 的对应参数

返回值：方程的解

源程序：

```
 int ext_euclid(int a,int b,int &x,int &y)  //求 gcd(a,b)=ax+by
{
    int t,d;
    if (b==0) {x=1;y=0;return a;}
    d=ext_euclid(b,a %b,x,y);
    t=x;
    x=y;
    y=t-a/b*y;
    return d;
}
void modular_equation(int a,int b,int n)
{
    int e,i,d;
    int x,y;
    d=ext_euclid(a,n,x,y);
    if (b%d>0)
       printf("No answer!\n");
    else
        {
       e=(x*(b/d))%n;
        for (i=0;i<d;i++)
            printf("The %dth answer is : %ld\n",i+1,(e+i*(n/d))%n);
        }
}
```

5.求解模线性方程组(中国余数定理)

语法：result=Modular_Expoent(int a,int b,int n);

参数：

B[]、W[]：a=B[] (mod W[]) 的对应参数

返回值：a 的值

注意：

其中 W[],B[]已知，W[i]>0 且 W[i]与 W[j]互质，求 a

源程序：

```
 int ext_euclid(int a,int b,int &x,int &y)  //求 gcd(a,b)=ax+by
{
    int t,d;
    if (b==0) {x=1;y=0;return a;}
    d=ext_euclid(b,a %b,x,y);
```

```
    t=x;
    x=y;
    y=t-a/b*y;
    return d;
}

int China(int B[],int W[],int k)
{
   int i;
    int d,x,y,a=0,m,n=1;
    for (i=0;i<k;i++)
        n*=W[i];
    for (i=0;i<k;i++)
       {
       m=n/W[i];
        d=ext_euclid(W[i],m,x,y);
        a=(a+y*m*B[i])%n;
        }
    if (a>0) return a;
    else return(a+n);
}
```

8.求距阵最大和

语法：result=maxsum2(int n);

参数：

a：距阵

n,m：距阵行列数

返回值：一维，二维距阵最大和

源程序：

```
 int a[101][101];
int maxsum(int a[],int n)//一维最大串
{
    int sum=-10000000,b=0;
    int i;
    for(i=0;i<n;i++)
        {
    if (b>0)
      b+=a[i];
    else
      b=a[i];
    if(b>sum)
      sum=b;
    }
  return sum;
```

```
}
int maxsum2(int m,int n)//二维最大串
{
  int sum = -10000000;
  int i,j,k,max;
  int* b = new int[n+1];
  for (i=0;i<m;i++)
    {
    for(k=0;k<n;k++)
      b[k]=a[i][k];
    max = maxsum(b,n);//第 i 列的一维最大串
    if(max>sum)
      sum=max;
    for(j=i+1;j<m;j++)
      {
      for (k=0;k<=n;k++)b[k]+=a[j][k];
      max = maxsum(b,n);//类似 maxsum，通过每列相加求二维最大串
      if(max>sum)sum=max;
      }
    }
  delete []b;
  return sum;
}
```

10.质因数分解

语法：result=int reduce(int prime[],int pn,int n,int rest[])

参数：

Prime[]：素数表，至少需要达到 sqrt(n)

pn：素数表的元素个数

N：待分解的数

Rest：分解结果，按照升序排列

返回值：分解因子个数

源程序：

```
 int reduce(int prime[],int pn,int n,int rest[])
{
int i,k=0;
for(i=0;i<pn;i++)
     {
     if (n==1) break;
     if (prime[i]*prime[i]>n) {rest[k++]=n;break;}
     while(n%prime[i]==0)
         {
         n/=prime[i];
         rest[k++]=prime[i];
```

```
        }
    }
return k;
}
```

11.高斯消元法解线性方程组

语法：gauss(int n,double ** a)

参数：

N：变量个数

Rest：变量系数行列式

源程序：

```
 void gauss(int n,double **a)
{
 int i, j, k;
 double client, temp = 0.0;

 for(k = 0; k < n - 1; k++)
  for(i = k + 1; i < n; i++)
  {
   client = a[i][k]/a[k][k];
   for(j = k + 1; j < n; j++)
    a[i][j] = a[i][j] - client * a[k][j];
   a[i][n] = a[j - 1][n] - client * a[k][n];
  }
 a[n - 1][n] = a[n - 1][n]/a[n - 1][n - 1];
 for(i = n - 2; i >= 0; i--)
 {
  for (j = i + 1; j < n; j++)
   temp += a[i][j] * a[j][n];
  a[i][n] = (a[i][n] - temp) / a[i][i];
 }
}

//打印
//for(i = 0; i < n; i++)
// printf("X%d = %lf\n", i + 1, a[i][n]);
```

五、图论

5.解欧拉图

语法：int Eular(int graph[8][8], int v, int *path)

参数：

graph：图，用邻接矩阵表示

v：图的顶点个数

path：D[i,j]表示从 i 到 j 的最短距离

注意：

此函数会删除图中的边

返回值：若找到欧拉回路则返回路径长度，否则返回-1

源程序：

```
 int Eular(int graph[8][8], int v, int *path)
{
    int start,a,b,count=0, deg,i;
    int s[1000], sp=0;//栈，大小可根据需要改变

    start=0;
    while(true)
    {
        a=start;
        s[sp++]=a;
        for(b=-1,i=0;i<v;i++) if (graph[i][a]&& a!=i) {b=i;break;}
        for(;b!=start&&b!=-1;)
        {
            s[sp++]=b;
            graph[a][b]=graph[b][a]=0;
            a=b;
            for(b=-1,i=0;i<v;i++) if (graph[i][a]&& a!=i) {b=i;break;}
        }
        if (b==-1) return(-1);//若找不到 Eular 回路返回-1
        s[sp++]=b;
        graph[a][b]=graph[b][a]=0;

        while(sp>0)
        {
            b=s[--sp];
            for(i=0,deg=0;i<v;i++) if (graph[i][b]==1) {deg++; break;}
            if (deg>0) break;
            path[count++]=b;
        }
        if (sp==0) return(count);
        start=b;
    }

}
```

八、高精度运算专题

1.专题函数说明

说明：

本栏为本专题所有程序的公用部分，调用本专题任何一个程序必须加上本栏的代码

input/print 为高精度数输入输出，调用格式为 input(hp HightPoint,"123456")/print(hp HighPoint)

本栏为纯 C++代码

源程序：

```
#include <iostream>
using namespace std;
#define maxsize 100
struct hp
{
   int len;
   int s[maxsize+1];
};

void input(hp &a,string str)
{
   int i;
   while(str[0]=='0' && str.size()!=1)
      str.erase(0,1);
   a.len=(int)str.size();
   for(i=1;i<=a.len;++i)
      a.s[i]=str[a.len-i]-48;
   for (i=a.len+1;i<=maxsize;++i)
      a.s[i]=0;
}
void print(const hp &y)
{
int i;
for(i=y.len;i>=1;i--)
   cout<<y.s[i];
cout<<endl;
}
```

2.高精度数比较

语法：int result=compare(const hp &a,const hp &b);

参数：

a,b：进行比较的高精度数字

返回值：比较结果，a>b 返回正数，a=b 返回 0，a<b 返回负数

源程序：

```
 int compare(const hp &a,const hp &b)
{
int len;
if(a.len>b.len)
   len=a.len;
else
   len=b.len;
while(len>0 && a.s[len]==b.s[len]) len--;
if(len==0)
   return 0;
else
   return a.s[len]-b.s[len];
}
```

5.高精度乘 10

语法：multiply10(hp &a);

参数：

a：进行乘法的高精度数字

返回值：返回结果到 a 中

源程序：

```
 void multiply10(hp &a)
{
int i;
for(i=a.len;i>=1;i--)
   a.s[i+1]=a.s[i];
a.s[1]=0;
a.len++;
while(a.len>1&&a.s[a.len]==0) a.len--;
}
```

8.高精度除单精度

语法：divide(const hp &a,int b,hp &c,int &d);

参数：

a：进行除法的高精度数字

返回值：返回商到 c 中，余数到 d 中

源程序：

```
 void divide(const hp &a,int b,hp &c,int &d)
{
int i,len;
for(i=1;i<=maxsize;i++) c.s[i]=0;
len=a.len;
```

```
d=0;
for(i=len;i>=1;i--)
   {
   d=d*10+a.s[i];
   c.s[i]=d/b;
   d%=b;
   }
while(len>1&&c.s[len]==0) len--;
c.len=len;
}
```

9.高精度除高精度

语法：divideh(const hp &a,const hp &b,hp &c,hp &d);

参数：

a，b：进行除法的高精度数字

返回值：返回商到 c 中，余数到 d 中

注意：

需要 compare、multiply10、subtract

源程序：

```
void divideh(const hp &a,const hp &b,hp &c,hp &d)
{
hp e;
int i,len;
for(i=1;i<=maxsize;i++)
   {
   c.s[i]=0;
   d.s[i]=0;
   }
len=a.len;
d.len=1;
for(i=len;i>=1;i--)
   {
   multiply10(d);
   d.s[1]=a.s[i];
   while(compare(d,b)>=0)
      {
      subtract(d,b,e);
      d=e;
      c.s[i]++;
      }
   }
while(len>1&&c.s[len]==0) len--;
c.len=len;
}
```

my own

单调队列

```
{
从队首 or 队尾删除
从队尾插入
}

type abc=record
     dat,num:longint;
     end;

var  q:array[1..1000] of abc;
     a:array[1..1000] of longint;
     tm,i,n,k,head,tail:longint;
     t:abc;

    procedure sort1(l,r: longint);
      var
         i,j,x,y: longint;
      begin
         i:=l;
         j:=r;
         x:=q[(l+r) div 2].dat;
         repeat
           while q[i].dat<x do
            inc(i);
           while x<q[j].dat do
            dec(j);
           if not(i>j) then
             begin
                y:=q[i].dat;
                q[i].dat:=q[j].dat;
                q[j].dat:=y;
                y:=q[i].num;
                q[i].num:=q[j].num;
                q[j].num:=y;
                inc(i);
                j:=j-1;
             end;
```

```
        until i>j;
        if l<j then
          sort1(l,j);
        if i<r then
          sort1(i,r);
      end;

    procedure sort2(l,r: longint);
      var
        i,j,x,y: longint;
      begin
        i:=l;
        j:=r;
        x:=q[(l+r) div 2].dat;
        repeat
          while q[i].dat>x do
           inc(i);
          while x>q[j].dat do
           dec(j);
          if not(i>j) then
            begin
              y:=q[i].dat;
              q[i].dat:=q[j].dat;
              q[j].dat:=y;
              y:=q[i].num;
              q[i].num:=q[j].num;
              q[j].num:=y;
              inc(i);
              j:=j-1;
            end;
        until i>j;
        if l<j then
          sort2(l,j);
        if i<r then
          sort2(i,r);
      end;

begin
readln(n,k);
for i:=1 to n do read(a[i]);

head:=k;  tail:=1;
for i:=1 to k do
 begin
 q[i].dat:=a[i];
```

```
 q[i].num:=i;
 end;
sort1(1,k);

for i:=k to n do
 begin
 tm:=tail;
 while (q[tail].dat<a[i]) and (tail<=head) do
  inc(tail);
 dec(tail);
 q[tail].dat:=a[i];
 q[tail].num:=i;
 while q[head].num<=i-k do dec(head);
 write(q[head].dat,' ');
 end;

writeln;//////////////////////////////////////////////////

head:=k;  tail:=1;
for i:=1 to k do
 begin
 q[i].dat:=a[i];
 q[i].num:=i;
 end;
sort2(1,k);

for i:=k to n do
 begin
 tm:=tail;
 while (q[tail].dat>a[i]) and (tail<=head) do
  inc(tail);
 dec(tail);
 q[tail].dat:=a[i];
 q[tail].num:=i;
 while q[head].num<=i-k do dec(head);
 write(q[head].dat,' ');
 end;

writeln;
end.
```

KMP 算法

```
//NOI 导刊  Mar.20th.2009
```

```
var t,p:string;
    lp,lt,i,q,k:longint;
    pf:array[1..1000] of longint;
    ok:boolean;

begin
ok:=false;
readln(t);
readln(p);
lt:=length(t);
lp:=length(p);

fillchar(pf,sizeof(pf),0);
pf[1]:=0;
k:=0;
for i:=2 to lp do
 begin
 while (k>0) and (p[k+1]<>p[i]) do
  k:=pf[k];
 if p[k+1]=p[i] then inc(k);
 pf[i]:=k;
 end;

q:=1;
for i:=1 to lt do
 begin
 while (q>0) and (t[i]<>p[q+1]) do
  q:=pf[q];
 if p[q+1]=t[i] then inc(q);
 if q=lp then
  begin
  ok:=true;
  writeln('START:',i-lp);
  end;
 end;

if not ok then writeln('404: NOT FOUND');

readln;
end.
```

多关键字排序

多关键字的快速排序
解决类似如下的问题：输入 n 组坐标（xi，yi），要求按 xi 的升序排序后，对于 xi=xj（j>i），yi~j 升序排列
即双关键字排序，先保证第一关键字的升序，在第一关键字一样的情况下保证第二关键字的升序

```
procedure qsort(v1,v1:longint);
var i,j,mid1,mid2,temp:longint;
begin
i:=v1; j:=v2;
mid1:=a[(i+j) shr 1,1];
mid2:=a[(i+j) shr 1,2];
while i<j do
begin
while (a[i,1]<mid1) or ((a[i,1]=mid1) and (a[i,2]<mid2)) do inc(i);
while (a[j,1]>mid1) or ((a[j,1]=mid1) and (a[j,2]>mid2)) do dec(j);
if i<=j then
begin
temp:=a[i,1]; a[i,1]:=a[j,1]; a[j,1]:=temp;
temp:=a[i,2]; a[i,2]:=a[j,2]; a[j,2]:=temp;
inc(i); dec(j);
end;
end;
if v1<j then qsort(v1,j);
if i<v2 then qsort(i,v2);
end;
```

多关键字的排序可以一次类推，比如再加一个第三关键字 Zi（保证降序）
即改为

```
while (a[i,1]<mid1) or ((a[i,1]=mid1) and (a[i,2]<mid2)) or((a[i,1]=mid1) and (a[i,2]=mid2) and (a[i,3]>mid3)) do inc(i);

while (a[j,1]>mid1) or ((a[j,1]=mid1) and (a[j,2]>mid2)) or((a[j,1]=mid1) and (a[j,2]=mid2) and (a[j,3]<mid3)) do dec(j);
```

次短路（SPFA）

{
思路：先用 SPFA 求一遍最短路，并将最短路上的每条边记录下来。
然后枚举删去最短路上的每一条边，再求一遍最短路。
删边之后求得的所有最短路中的的最小值即次短路

求次小生成树？方法同上
先求最小生成树，然后枚举删除其中的每一条边再求一遍。

所有求得的.....中的最小值即次小生成树

注意：
因为最短路\最小生成树可能有多个，
所以可能出现求得的次短路=最短路、次小生成树=最小生成树这样的情况
}

```
var path,q,d:array[1..1000] of longint;
    a:array[1..1000,1..1000] of longint;
    visited:array[1..1000] of boolean;
    tm1,tm2,min,tmp,head,tail,s,t,n,m,dt,i,j:longint;

procedure spfa2;
begin
fillchar(d,sizeof(d),127 div 3);
fillchar(visited,sizeof(visited),false);
fillchar(q,sizeof(q),0);
d[s]:=0;
head:=0;
q[1]:=s;
visited[1]:=true;
tail:=1;
while head<tail do
 begin
 inc(head);
 visited[q[head]]:=false;
 for i:=1 to n do
  begin
  if (a[q[head],i]>0) and (d[q[head]]+a[q[head],i]<d[i]) then
   begin
   d[i]:=d[q[head]]+a[q[head],i];
   if not visited[i] then
    begin
    inc(tail);
    q[tail]:=i;
    visited[i]:=true;
    end;
   end;
  end;
 end;
end;

procedure spfa1;
begin
fillchar(d,sizeof(d),127 div 3);
fillchar(visited,sizeof(visited),false);
```

```
fillchar(path,sizeof(path),0);
fillchar(q,sizeof(q),0);
d[s]:=0;
head:=0;
q[1]:=s;
visited[1]:=true;
tail:=1;
while head<tail do
 begin
 inc(head);
 visited[q[head]]:=false;
 for i:=1 to n do
  begin
  if (a[q[head],i]>0) and (d[q[head]]+a[q[head],i]<d[i]) then
   begin
   d[i]:=d[q[head]]+a[q[head],i];
   path[i]:=q[head];
   if not visited[i] then
    begin
    inc(tail);
    q[tail]:=i;
    visited[i]:=true;
    end;
   end;
  end;
 end;
end;

begin
fillchar(a,sizeof(a),0);
{readln(s,t);
readln(n);
for i:=1 to n do
 for j:=1 to n do
  begin
  read(dt);
  a[i,j]:=dt;
  a[j,i]:=dt;
  end;
}
readln(n,m,s,t);
for i:=1 to m do
 begin
 readln(tm1,tm2,dt);
 a[tm1,tm2]:=dt;
```

```
        a[tm2,tm1]:=dt;
        end;

       SPFA1;
       dt:=d[t];
       writeln(dt);
       {
       for i:=1 to n do
        write(d[i],' ');
       writeln;
       }
       //for i:=1 to n do
       // begin
       tmp:=t;
       min:=maxlongint;
       //write(tmp,' ');
       while path[tmp]<>0 do
        begin
        //write(path[tmp],' ');
        //writeln(tmp,'--',path[tmp]);
        tm1:=a[tmp,path[tmp]];
        tm2:=a[path[tmp],tmp];
        a[tmp,path[tmp]]:=0;
        a[path[tmp],tmp]:=0;
        SPFA2;
        if (d[t]>dt) and (d[t]<min) then min:=d[t];
        a[tmp,path[tmp]]:=tm1;
        a[path[tmp],tmp]:=tm2;
        tmp:=path[tmp];
        end;
       writeln(min);
       // end;

       close(input);
       {
       close(output);
       }
   end.
```

快速幂

```
       var n,m,p,ans:int64;

       function calc(a,b,c:int64):int64;
```

```
var t:int64;
begin
if b=0 then exit(1)
else if b=1 then exit(a mod c)
else
 begin
 t:=calc(a,b shr 1,c);
 t:=(t*t) mod c;
 if b mod 2>0 then
  t:=(t*a) mod c;
 exit(t);
 end;
end;

begin
readln(n,m,p);
ans:=m mod p;
dec(n);
dec(m);

{
if n mod 2<>0 then
 begin
 ans:=ans*(m mod p);
 dec(n);
 end;
}
ans:=ans*calc(m,n,p);
ans:=ans mod p;
writeln(ans);
 end.
```

Huffman

```
//合并果子
//n 个结点，构造一棵最优二叉树，输出总权值
var a:array[1..50000] of longint;
    n,x,y,i,j,k,sum:longint;

begin
readln(n);
for i:=1 to n do read(a[i]);

qsort(1,n);
```

```
        i:=1;

        while i<n do
         begin
         x:=a[i];   y:=a[i+1];
         sum:=sum+x+y;
         i:=i+2;
         j:=i;
         while (a[j]<(x+y)) and (j<n+1) do inc(j);
         for k:=n downto j do
          a[k+1]:=a[k];
         a[j]:=x+y;
         inc(n);
         end;

        writeln(sum);
        readln;
    end.
```

求 n 以内质数

```
//1.开一个大的 bool 型数组 prime[]，大小就是 n+1 就可以了.
//  先把所有的下标为奇数的标为 true,下标为偶数的标为 false.
//2.然后：
//    for( i=3; i<=sqrt(n); i+=2 )
//    {   if(prime)
//        for( j=i+i; j<=n; j+=i ) prime[j]=false;
//    }
//3.最后输出 bool 数组中的值为 true 的单元的下标，就是所求的 n 以内的素数了。
//
//原理很简单，就是当 i 是质(素)数的时候，i 的所有的倍数必然是合数。
//如果 i 已经被判断不是质数了，那么再找到 i 后面的质数来把这个质数的倍数筛掉。
```

求组合数

```
        {
        算法：
        prime(x)：求出 1~x 之间的质数，并加到 b[i].dat 中
        calcprime(x)：求出 1~x 之间的质数，并计入 f[i]
        calc(x)：对 x!分解质因数，计算出 x!中各质因数的使用次数。
                即 Σ((b[i].dat)^(b[i].time))=x!
```

因为 C(n,m):=(n!)/(m!*(n-m)!)
所以对 n!分解质因数，计算出 n!中各质因数的使用次数，计入 a
再分别对 m!、(n-m)!进行类似操作，并在 a 数组中减去相应的质因数的使用次数
最后将 a 中的质数按各自的幂相乘即可。
}

```
type abc=record
     dat,time:longint;
     end;

var f:array[1..10000] of boolean;
    a,b:array[1..10000] of abc;
    ans,fm,i,j,n,nm,m,tn:longint;

procedure prime(x:longint);
begin
i:=2;
nm:=0;
while i<=x do
 begin
 if f[i] then
  begin
  inc(nm);
  b[nm].dat:=i;
  end;
 inc(i);
 end;
end;

procedure calc(x:longint);
var t:longint;
begin
for i:=1 to nm do
 begin
 t:=x;
 while t<>0 do
  begin
  t:=t div b[i].dat;
  inc(b[i].time,t);
  end;
 end;
end;

procedure calcprime(x:longint);
var tn,i,j:longint;
begin
```

```
tn:=trunc(sqrt(n))+1;
fillchar(f,sizeof(f),true);
f[1]:=false;
for i:=1 to n do
 if (i mod 2=0) or (i mod 5=0) then
  f[i]:=false;
f[2]:=true;
f[5]:=true;
for i:=3 to tn do
 if f[i] then
  begin
  j:=i+i;
  while j<=n do
   begin
   f[j]:=false;
   inc(j,i);
   end;
  end;
end;

begin
readln(n,m);
//C(n,m)
fillchar(f,sizeof(f),true);
calcprime(n);
{
f[1]:=false;
tn:=trunc(sqrt(n))+1;
for i:=2 to tn do
 for j:=2 to tn do
  f[i*j]:=false;
}
fillchar(b,sizeof(b),0);
prime(n);
calc(n);
fm:=nm;
a:=b;

fillchar(b,sizeof(b),0);
prime(m);
calc(m);
for i:=1 to nm do
 dec(a[i].time,b[i].time);

fillchar(b,sizeof(b),0);
```

```
prime(n-m);
calc(n-m);
for i:=1 to nm do
 dec(a[i].time,b[i].time);

ans:=1;
for i:=1 to fm do
 for j:=1 to a[i].time do
  ans:=ans*a[i].dat;
writeln(ans);
readln;
end.
```

二分答案模版

1. 前言：

二分答案顾名思义就是二分出答案来验证
通俗一点来说就是在[1,∞]上
每次取中间点，猜答案..
把答案作为已知条件
判断答案是否满足题意..
从而求证出所需解的过程

2. 方法：

```
var
l,r,m:longint;
ans:longint;
begin
  l:=1; r:=n;
  ans:=0;
  while l<=r do
    begin
      m:=(l+r) shr 1;
      if pd(m) then begin ans:=m; r:=m-1; end
else l:=m+1
    end;
     writeln(ans);
end;
```

最优子矩阵

{
思路：

将高度为 k，长度从 i 到 j 的一个子矩阵压缩成一个 i~j 的序列，
将高度为 k 的一列压成一个点，该点的值为这 k 个数的和，
这样就得到了一个 i~j 的序列。
对这个序列求连续最大和，这个值即这个矩阵的最优子矩阵

注意：sum[i,j]表示第 j 列上从上到下第 1 个数到第 j 个数的和
那么第 p 个到第 q 个数的和就是 sum[q,j]-sum[p-1,j]
这种优化方法很有用！
}

```
var n,m,i,j,ans,mx,k:longint;
    f,tmp:array[1..1000] of longint;
    a:array[1..1000,1..1000] of longint;
    sum:array[0..1000,1..1000] of longint;

function max(a,b:longint):longint;
begin
if a>b then exit(a) else exit(b);
end;

begin
assign(input,'zz.in');
reset(input);

readln(n,m);
for i:=1 to n do
 for j:=1 to m do
  read(a[i,j]);

fillchar(sum,sizeof(sum),0);
for i:=1 to n do
 for j:=1 to m do
  sum[i,j]:=sum[i-1,j]+a[i,j];

ans:=0;

for i:=1 to n do
 for j:=i to n do
  begin
  for k:=1 to m do
   tmp[k]:=sum[j,k]-sum[i-1,k];
  fillchar(f,sizeof(f),0);
  f[1]:=tmp[1];
  for k:=2 to m do
```

```
   f[k]:=max(f[k-1]+tmp[k],tmp[k]);
  mx:=0;
  for k:=1 to m do
   if f[k]>mx then mx:=f[k];
  if mx>ans then ans:=mx;
  end;

writeln(ans);

close(input);
end.
```

连续最大和

```
{
f[i]表示以 i 结尾的连续最大和
f[i]:=max(f[i-1]+a[i],a[i])
ans:=max(f[i])
}

var a,f:array[1..1000] of longint;
    i,n,mx:longint;

function max(a,b:longint):longint;
begin
if a>b then exit(a) else exit(b);
end;

begin
readln(n);
for i:=1 to n do
 read(a[i]);

fillchar(f,sizeof(f),0);
f[1]:=a[1];

for i:=2 to n do
 f[i]:=max(f[i-1]+a[i],a[i]);

mx:=0;
for i:=1 to n do
 if f[i]>mx then mx:=f[i];

writeln;
```

```
writeln(mx);
end.
```

RMQ

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

#define M 100010
#define MAXN 500
#define MAXM 500
int dp[M][18];
/*
   求区间内的最值
*一维 RMQ ST 算法
*构造 RMQ 数组 makermq(int n,int b[]) O(nlog(n))的算法复杂度
*dp[i][j] 表示从 i 到 i+2^j -1 中最小的一个值(从 i 开始持续 2^j 个数)
*dp[i][j]=min{dp[i][j-1],dp[i+2^(j-1)][j-1]}
*查询 RMQ rmq(int s,int v)
*将 s-v 分成两个 2^k 的区间
*即 k=(int)log2(s-v+1)
*查询结果应该为 min(dp[s][k],dp[v-2^k+1][k])
*/

void makermq(int n,int b[])
{
    int i,j;
    for(i=1;i<=n;i++)
        dp[i][0]=b[i];
    for(j=1;(1<<j)<=n;j++)
        for(i=1;i+(1<<j)-1<=n;i++)
            dp[i][j]=min(dp[i][j-1],dp[i+(1<<(j-1))][j-1]);
}
int rmq(int s,int v)
{
    int k=(int)(log((v-s+1)*1.0)/log(2.0));
    return min(dp[s][k],dp[v-(1<<k)+1][k]);
}

void makeRmqIndex(int n,int b[]) //返回最小值对应的下标
{
    int i,j;
```

```
    for(i=1;i<=n;i++)
        dp[i][0]=i;
    for(j=1;(1<<j)<=n;j++)
        for(i=1;i+(1<<j)-1<=n;i++)
            dp[i][j]=b[dp[i][j-1]] < b[dp[i+(1<<(j-1))][j-1]]? dp[i][j-1]:dp[i+(1<<(j-1))][j-1];
}
int rmqIndex(int s,int v,int b[])
{
    int k=(int)(log((v-s+1)*1.0)/log(2.0));
    return b[dp[s][k]]<b[dp[v-(1<<k)+1][k]]? dp[s][k]:dp[v-(1<<k)+1][k];
}

int main()
{
    int a[]={-1,3,4,5,7,8,9,0,3,4,5};
    //返回下标
    makeRmqIndex(sizeof(a)/sizeof(a[0]),a);
    cout<<rmqIndex(1,10,a)<<endl;
    cout<<rmqIndex(5,10,a)<<endl;
    //返回最小值
    makermq(sizeof(a)/sizeof(a[0]),a);
    cout<<rmq(1,10)<<endl;
    cout<<rmq(5,10)<<endl;
    return 0;
}
```

ACM 会用到的一点数学知识

1.费马小定理：a^p mod p=a（p 为素数，且 a 不是 p 的倍数）

2.数 n 的约数个数：

n 分解因数为 p1^s1*p2^s2*……pm^sm

则约数个数为(s1+1)*(s2+1)*……*(sm+1)

3.Fibonacci 数通项公式：Fn=round((1+√5)/2)^n/√5

4.Catalan 数通项公式：Cn=C(2n-2,n-1)/n

递归式：Cn=∑Ci*C(n-i) (i=1..n-1,C1=C2=1)

5.第二类 Stirling 数：S(n,k)表示 n 个元素的集合拆分成 k 部分的数

S(n,k)=S(n-1,k-1)+k*S(n-1,k)

6.整数分拆：P(n,k)-整数 n 分成 k 部分的数

P(n,k)=P(n-1,k-1)+P(n-k,k)

7.方程 x1+x2+……+xk=n (xi>=0)的解的个数：C(n+k-1,k-1)

方程 x1+x2+……+xk=n (xi>0)的解的个数：C(n-1,k-1)

计数排序

```
#include <iostream>
#include <cstring>
using namespace std;
int a[10000],b[10000],c[10000],n,i;

int main()
{
 memset(c,0,sizeof(c));
 memset(b,0,sizeof(b));

 cin>>n;
 int mx=0,mn=99999;
 for (i=1;i<=n;i++)
 {
     cin>>a[i];
     c[a[i]]++;
     if (a[i]>mx) mx=a[i];
     if (a[i]<mn) mn=a[i];
 }
 for (i=mn+1;i<=mx;i++)
     c[i]=c[i]+c[i-1];

 for (i=1;i<=n;i++)
     cout<<c[i]<<" ";
 cout<<endl;

 for (i=1;i<=n;i++)
 {
     b[c[a[i]]]=a[i];
     c[a[i]]--;
 }
 for (i=1;i<=n;i++)
     cout<<b[i]<<" ";
 cout<<endl;
}
```

高精度加法

```
#include <iostream>
#include <cstring>
using namespace std;
```

```
int main()
{
 string sa,sb;
 int la,lb,i,l;
 int a[100],b[100],c[100];
 memset(c,0,sizeof(c));
 memset(a,0,sizeof(a));
 memset(b,0,sizeof(b));

   cin>>sa;
   cin>>sb;
 la=sa.length();
 lb=sb.length();
 for (i=1;i<=la;i++)
     a[i]=sa[la-i]-'0';
 for (i=1;i<=lb;i++)
     b[i]=sb[lb-i]-'0';
 if (la>=lb) l=la; else l=lb;

 for(i=1;i<=l;i++)
 {
     c[i]=c[i]+a[i]+b[i];
     c[i+1]=c[i+1]+c[i]/10;
     c[i]=c[i]%10;
 }

 if (c[l+1]!=0) l++;
 for (i=l;i>=1;i--)
     cout<<c[i];
   cout<<endl;
 return 0;
}
```

高精度减法

```
#include <iostream>
#include <cstring>
using namespace std;

int main()
{
 int a[100],b[100],c[100],la,lb,l,i,tmp,fh;
 string sa,sb,ts;
```

```
memset(a,0,sizeof(a));
memset(b,0,sizeof(b));
memset(c,0,sizeof(c));

cin>>sa;
cin>>sb;
la=sa.length();
lb=sb.length();

  fh=1;
if (sa==sb)
{
    cout<<0<<endl;
    return 0;
}
else if (((sa<sb)&&(la==lb))||(la<lb))
{
    fh=0;
    ts=sb;
    sb=sa;
    sa=ts;
}

la=sa.length();
lb=sb.length();
for (i=1;i<=la;i++)
   a[i]=sa[la-i]-'0';
for (i=1;i<=lb;i++)
   b[i]=sb[lb-i]-'0';

l=la;
for (i=1;i<=l;i++)
{
    if (a[i]<b[i])
    {
        a[i+1]--;
        a[i]=a[i]+10;
    }
    a[i]=a[i]-b[i];
}

  if (fh==0) cout<<'-';
  while (a[l]==0) l--;
for (i=l;i>=1;i--)
   cout<<a[i];
```

```
    cout<<endl;
    return 0;
}
```

高精度乘法

```
#include <iostream>
#include <cstring>
using namespace std;

int main()
{
 int a[100],b[100],c[100],la,lb,l,i,j;
 string sa,sb;
 memset(a,0,sizeof(a));
 memset(b,0,sizeof(b));
 memset(c,0,sizeof(c));

 cin>>sa;
 cin>>sb;
 la=sa.length();
 lb=sb.length();
 for (i=1;i<=la;i++)
     a[i]=sa[la-i]-'0';
 for (i=1;i<=lb;i++)
     b[i]=sb[lb-i]-'0';

   for (i=1;i<=la;i++)
   {
     for (j=1;j<=lb;j++)
     {
          c[i+j-1]=c[i+j-1]+a[i]*b[j];
          c[i+j]=c[i+j]+c[i+j-1]/10;
          c[i+j-1]=c[i+j-1]%10;
     }
   }

   l=la+lb;
   while ((l>1)&&(c[l]==0)) l--;
 for (i=l;i>=1;i--)
     cout<<c[i];
 cout<<endl;
 return 0;
}
```

归并排序

```
#include <iostream>
#include <cstring>
using namespace std;
int n;   int ans=0;
int a[1000];

void merge(int l,int t,int r)
{
 int tl[1000],tr[1000];
 int i;
 for (i=1;i<=t-l+1;i++)
     tl[i]=a[l+i-1];
 for (i=1;i<=r-(t+1)+1;i++)
     tr[i]=a[t+1+i-1];
 int li=1;    int ri=1;
 tl[t-l+2]=99999;    tr[r-t+1]=99999; //哨兵:最顶端的设无穷大,防越界
 for (i=l;i<=r;i++)
 {
     if (tl[li]<=tr[ri])
     {
         a[i]=tl[li];
         li++;
     }
     else
     {
         a[i]=tr[ri];
         ri++;
         ans=ans+(t-l+1-li+1);
     }
 }
}

void mergesort(int l,int r)
{
 if (l==r) return;
 else if (r==l+1)
 {
     if (a[l]>a[r])
     {
         int tm=a[l];
         a[l]=a[r];
         a[r]=tm;
```

```
                ans++;
            }
        }
        else if (r>l)
        {
            int tmp=(l+r)/2;
            mergesort(l,tmp);
            mergesort(tmp+1,r);
            merge(l,tmp,r);
        }
    }

    int main()
    {
        int i;
        memset(a,0,sizeof(a));

        cin>>n;
        for (i=1;i<=n;i++)
            cin>>a[i];

        mergesort(1,n);

        for (i=1;i<=n;i++)
            cout<<a[i]<<" ";
        cout<<endl;
        cout<<ans<<endl;                    //求逆序对的数目
}
```

位运算

a shl b 的值实际上就是 a 乘以 2 的 b 次方，因为在二进制数后添一个 0 就相当于该数乘 2。

a shr b 表示二进制右移 b 位（去掉末 b 位），相当于 a 除以 2 的 b 次方（取整）。

有时我们的程序需要一个规模不大的 Hash 表来记录状态。比如，做数独时我们需要 27 个 Hash 表来统计每一行、每一列和每一个小九宫格里已经有哪些数了。此时，我们可以用 27 个小于 2^9 的整数进行记录。例如，一个只填了 2 和 5 的小九宫格就用数字 18 表示（二进制为 000010010），而某一行的状态为 511 则表示这一行已经填满。需要改变状态时我们不需要把这个数转成二进制修改后再转回去，而是直接进行位操作。在搜索时，把状态表示成整数可以更好地进行判重等操作。

去掉最后一位	(101101->10110)	x shr 1
在最后加一个 0	(101101->1011010)	x shl 1
在最后加一个 1	(101101->1011011)	x shl 1+1
把最后一位变成 1	(101100->101101)	x or 1
把最后一位变成 0	(101101->101100)	x or 1-1
最后一位取反	(101101->101100)	x xor 1
把右数第 k 位变成 1	(101001->101101,k=3)	x or (1 shl (k-1))

把右数第 k 位变成 0	(101101->101001,k=3)	x and not (1 shl (k-1))
右数第 k 位取反	(101001->101101,k=3)	x xor (1 shl (k-1))
取末三位	(1101101->101)	x and 7
取末 k 位	(1101101->1101,k=5)	x and (1 shl k-1)
取右数第 k 位	(1101101->1,k=4)	x shr (k-1) and 1
把末 k 位变成 1	(101001->101111,k=4)	x or (1 shl k-1)
末 k 位取反	(101001->100110,k=4)	x xor (1 shl k-1)
把右边连续的 1 变成 0	(100101111->100100000)	x and (x+1)
把右起第一个 0 变成 1	(100101111->100111111)	x or (x+1)
把右边连续的 0 变成 1	(11011000->11011111)	x or (x-1)
取右边连续的 1	(100101111->1111)	(x xor (x+1)) shr 1
去掉右起第一个 1 的左边	(100101000->1000)	x and (x xor (x-1))

逆矩阵

```
# include "stdio.h"
# define M  3
void main (  )
 {
  float MAT[M][2*M];
  float MAT1[M][M];
  float t;
  int i,j,k,l;
/*********************************************/
   /*对矩阵进行初始化*/
  for(i=0;i<M;i++)
    for(j=0;j<2*M;j++)
      MAT1[i][j]='\0';

  for(i=0;i<M;i++)
     for(j=0;j<2*M;j++)
     MAT[i][j]='\0';

    /*对 MAT1 矩阵赋初值  */
  for(i=0;i<M;i++)
    for (j=0;j<M;j++)
      scanf("%f",&MAT1[i][j]);

  /*打印目标矩阵?*/
  printf("原矩阵为：\n");
  for (i=0;i<M;i++)
    {
     for (j=0;j<M;j++)
printf("%5.2f",MAT1[i][j]);
```

```
     printf("\n");
     }
/*****************************************/
    /*对 MAT1 矩阵进行扩展,MAT1 矩阵添加单位阵，由 M*M 变成 2M*2M 矩阵  */
  for(i=0;i<M;i++)
    for(j=0;j<2*M;j++)
      if (j<M)  MAT[i][j]=MAT1[i][j];
     else if (j==M+i)  MAT[i][j]=1;
   else MAT[i][j]=0;

    /*对 M 矩阵进行变换，使得前半部分矩阵成为单位阵，则  */
    /*后半部分矩阵即为所求矩阵逆阵  */
  for(i=0;i<M;i++)
   {
    /*对第 i 行进行归一化    */
     for (j=0;j<2*M;j++)
     for(k=i+1;k<M;k++)
MAT[i][j]=MAT[i][j]+MAT[k][j];
     t=MAT[i][i];
     for(j=i;j<2*M;j++)
MAT[i][j]=MAT[i][j]/t;

     /*对矩阵进行行变换，使得第 i 列只有一个元素不为零，且为 1*/
     for(k=0;k<M;k++)
if(k!=i)
  {
   t=MAT[i][k];
   for (l=i;l<2*M;l++)
   MAT[k][l]=MAT[k][l]-MAT[i][l]*t;
  }
     }

 /*将后半部分矩阵即所求矩阵逆阵存入 MAT2 矩阵。*/
  for(i=0;i<M;i++)
   {

    for(j=0;j<M;j++)
      MAT1[i][j]=MAT[i][j+M];
    printf("\n");

   }
/*****************************************/
   /*输出所求的逆阵*/
   printf("逆阵为：\n");
   for(i=0;i<M;i++)
```

```
        {

          for(j=0;j<M;j++)
          printf("%5.2f",MAT1[i][j]);
          printf("\n");
        }
}
```

行列式

```
#include<stdio.h>
int main()
{
    int n,i,j,k,max;
    double u[20][20],temp1,temp2,rult;
    while(scanf("%d",&n)!=EOF)
    {
        for(i=0;i<n;i++)
            for(j=0;j<n;j++)
                scanf("%lf",&u[j]);
        // 矩阵变换上三角
        for(i=0;i<n-1;i++)
        {
            max=i;
            for(j=i+1;j<n;j++)
            {
                temp1=u<0 ? -u : u;
                temp2=(u[j]<0 ?-u[j]:u[j]);
                if(temp2>temp1)max=j;
            }
            if(max!=i)
                for(j=i;j<n;j++)
                {
                   temp1=u[max][j];
                   u[max][j]=u[j];
                   u[j]=temp1;
                }
            for(j=i+1;j<n;j++)
            {
                temp1=u[j]/u;
                for(k=i;k<=n;k++)
                    u[j][k]=u[j][k]-u[k]*temp1;
            }
        }
```

```
        //依次求解
        for(rult=1,i=0;i<n;i++)
           rult*=u;
        printf("%.4lf",rult);
    }
    return 0;
}
```

floyd 求最小环

利用 Floyed 算法求最小环

朴素算法：

令 e(u,v)表示 u 和 v 之间的连边，再令 min(u,v)表示删除 u 和 v 之间的连边之后 u 和 v 之间的最短路。

最小环则是 min(u,v) + e(u,v)。时间复杂度是 O(E*V^2)。

改进算法：

在进行 floyed 算法的同时，顺便算出最小环。

（注：g[i][j]=i,j 之间的边长）

```
dist:=g;
for k:=1 to n do
begin
    for i:=1 to k-1 do
        for j:=i+1 to k-1 do
            answer:=min(answer,dist[i][j]+g[i][k]+g[k][j]);
    for i:=1 to n do
        for j:=1 to n do
            dist[i][j]:=min(dist[i][j],dist[i][k]+dist[k][j]);
end;
```

只允许经过1..k-1点，不经过k

可以经过k点（标准的floyd过程）

other

函数：cout<<fixed<<setprecision(2)<<x; 功能：输出 x,精确刀小数点后两位

```
#include <iostream>
#include <iomanip>
using namespace std;
int main()
{
    int x;
    cin>>x;
    cout<<fixed<<setprecision(2)<<x;
    return 0;
```

```
}
```

函数：pow(a,b+0.0); 功能：计算出 a 的 b 次方

```
#include <iostream>
#include <cmath>
using namespace std;
int main()
{
    int x,y;
    cin>>x>>y;
    cout<<pow(x,y+0.0);
    return 0;
}
```

备注： b 后面要加 0.0，因为必须是实数型。

函数：itoa (a , b , 10) ; 功能：把 a 的值的 10 进制放到字符串 b 里；

```
#include <iostream>
using namespace std;
int main()
{
    char str[11];
    int i;
    cin>>i;
    itoa(i,str,10);
    cout<<str;
    return 0;
}
```

函数：atoi (a , b) ;

功能：取串 a 的数字部分；这个函数我觉得特别不错，其不错的原因是：1.如果 a 到某一位以后不是数字了，那么就只取到这个串之前的数。2.如果 a 串里的的数值是负的，那么取到的值也是负的。3.比如你想取 a 的第 i 位以后的数值，那么就可以 b=atoi(a+i);

```
#include <iostream>
using namespace std;
int main()
{
    char str[11];
    int i;
    cin>>str;
    i=atoi(str);
    cout<<i;
    return 0;
}
```

Addition:线段树专题总结

(1)单点更新

常用模板：

```
Struct treetype
{
    int l,int r,int o;
}t[3000];                  //如果要对长度为 n 的区间建线段树，那么 t 数组至少要打到 3*n
```

建树：

```
void build(int l,int r,int o)
{
    if (o>num) num=o;          //num:计数用，记录线段树上已有多少节点
    t[o].l=l;   t[o].r=r;
    if (l==r)
        t[o].dat=a[l];         //更新只包含一个点的区间（即树上的叶子节点）
    else
    {
        int mid=(l+r)/2;
        build(l,mid,2*o);
        build(mid+1,r,2*o+1);
        t[o].dat=max/min/sum(t[2*o].dat,t[2*o+1].dat);
                    //根据要求而定
    }
}
```

更新节点：

1. 对于求区间和的问题：

   更新：令 a[x]=m;

```
        void update(int l,int r,int o)      //a[x]=m;
        {
         t[o].l=l;   t[o].r=r;
            if ((l==x)&&(r==x))                //找到点节点：更新点
                t[o].dat=m;
            else
            {
                int mid=(l+r)/2;
                if (x<=mid)           //否则：对左 or 右区间进行处理
                {
                    int tmp=t[2*o].dat;
                    update(l,mid,2*o);
                    t[o].dat+=t[2*o].dat-tmp;
                }
                else
                {
```

```
                int tmp=t[2*o+1].dat;
                update(mid+1,r,2*o+1);
                t[o].dat+=t[2*o+1].dat-tmp;
            }
        }
    }
```

2. 对于求区间最值的问题：

更新：令 a[tx]=ty;

```
void update(int l,int r,int o)
{
    if (o>num) return;          //防止出现 o 不停增加的死循环
                                        （其实加不加也无所谓-_-||）
    if ((l==tx)&&(r==tx))           //找到点节点：更新点
        t[o].dat=ty;
    else
    {
        int mid=(l+r)/2;
        if (tx<=mid)            //否则：对左 or 右区间进行处理
        {
            update(l,mid,2*o);
            t[o].dat=max(t[o].dat,t[2*o].dat);
        }
        else
        {
            update(mid+1,r,2*o+1);
            t[o].dat=max(t[o].dat,t[2*o+1].dat);
        }
    }
}
```

求区间上的某种性质：

1. 求区间和

```
int query_sum(int l,int r,int o)
{
    if (l>r)                //不可行的情况
        return 0;
    else if (l==r)
        return a[l];
    else
    {
        int tl=t[o].l,tr=t[o].r;
        if ((l==tl)&&(r==tr))       //正好命中线段树上的一整块区间，直接返回值即可
            return t[o].dat;
        else
        {
```

```
            int mid=(tl+tr)/2;      //否则拆开：l...mid 和 mid+1...r
            if (r<=mid)             //注意红字部分！
                return (query_sum(l,r,2*o));
            else if (l>mid)
                return (query_sum(l,r,2*o+1));
            else
                return (query_sum(l,mid,2*o)+query_sum(mid+1,r,2*o+1));
        }
    }
}
```

2. 求区间最值（以最大值为例）：

```
int query_max(int l,int r,int o)
{
    if (o>num) return 0;            //不可行的情况
    int tl=t[o].l,tr=t[o].r;
    if ((l==tl)&&(r==tr))       //正好命中线段树上的一整块区间，直接返回值即可
        return t[o].dat;
    else
    {
        int mid=(tl+tr)/2;          //否则拆开：l...mid 和 mid+1...r
        if (r<=mid)                     //注意红字部分！
            return (query_max(l,r,2*o));
        else if (l>mid)
            return (query_max(l,r,2*o+1));
        else if ((l<=mid)&&(mid<=r))
        {
            int tm1=query_max(l,mid,2*o);
            int tm2=query_max(mid+1,r,2*o+1);
            return max(tm1,tm2);
        }
    }
}
```

主程序中：

```
cin>>n;
for i:=1 to n do cin>>a[i];
build(1,n,1);                               //建树

for i:=1 to m do
{
     cin>>tx>>ty;
     a[tx]=ty;
     update(1,n,1);                         //单点更新
}
```

```
cin>>ml>>mr;
ans1=query_sum(ml,mr,1);                //求区间和
ans2=query_max(ml,mr,1);                //求区间最值
```

备注：模板中的除法操作还可以用位运算优化：int mid=(l+r)>>1;

附录：网上的一段模板（hdu 1166）

```
#include<stdio.h>
#define MAX 50000+10
int head[MAX];
struct node
{
    int l,r;
    int value;
}tree[3*MAX];

void build(int l,int r,int v)           //建树
{
    tree[v].l=l;
    tree[v].r=r;
    if(l==r)
    {
        tree[v].value=head[l];
        return ;
    }
    int mid=(l+r)>>1;
    build(l,mid,v*2);
    build(mid+1,r,v*2+1);
    tree[v].value=tree[v+v].value+tree[v+v+1].value;
}

int queue(int a,int b,int v)            //求和
{
    if(tree[v].l==a&&tree[v].r==b)
    {
        return tree[v].value;
    }
    int mid=(tree[v].l+tree[v].r)>>1;
    if(b<=mid) return queue(a,b,v+v);
    else if(a>mid) return queue(a,b,v+v+1);
    else return queue(a,mid,v*2)+queue(mid+1,b,v*2+1);
}

void update(int a,int b,int v)      //更新: head[a]+=b;
{
```

```
    tree[v].value+=b;
    if(tree[v].l==tree[v].r)
    {
        return ;
    }
    int mid=(tree[v].l+tree[v].r)>>1;
    if(a<=mid) update(a,b,v*2);
    else update(a,b,v*2+1);
}

int main()
{
    int T,N;
    char str[10];
    int a,b;
    while(scanf("%d",&T)>0)
    {
        for(int h=0;h<T;h++)
        {
           scanf("%d",&N);
           for(int i=1;i<=N;i++)
           {
               scanf("%d",&head[i]);
           }
           build(1,N,1);
           printf("Case %d:\n",h+1);
           while(scanf("%s",str)>0)
           {
               if(str[0]=='A')
               {
                   scanf("%d %d",&a,&b);
                   update(a,b,1);
               }
               else if(str[0]=='S')
               {
                    scanf("%d %d",&a,&b);
                    update(a,-b,1);
               }
               else if(str[0]=='Q')
               {
                   scanf("%d %d",&a,&b);
                    int ans=queue(a,b,1);
                    printf("%d\n",ans);
               }
               else
```

```
                {
                    break;
                }
            }

        }
    }
}
```

2.成段更新

```
struct
{
    int l,r;
    __int64 dat;
}t[300010];

char ch;
__int64 ml,mr,md;
__int64 add[300010];
```

建树：

```
void build(long l,long r,long long o)
{
    if (o>num) num=o;
    t[o].l=l;   t[o].r=r;
    if (l==r)
        t[o].dat=a[l];
    else
    {
        long mid=(l+r)/2;
        build(l,mid,2*o);
        build(mid+1,r,2*o+1);
        t[o].dat=t[2*o].dat+t[2*o+1].dat;
    }
}
```

更新：cin>>ml>>mr>>md; update(ml,mr,1); ->令 a[ml..mr]+=md

```
void update(long l,long r,long long o)
{
    if (o>num) return;
    long tl=t[o].l,tr=t[o].r;
    if ((tl==l)&&(tr==r))
    {
        t[o].dat+=(tr-tl+1)*md;
```

```
            add[o]+=md;
            return;
        }
        else
        {
            long mid=(tl+tr)/2;
            //if (tl==tr) return;
            push_down(o,tl,mid,tr);
            if (r<=mid)
                update(l,r,2*o);
            else if (l>mid)
                update(l,r,2*o+1);
            else
            {
                update(l,mid,2*o);
                update(mid+1,r,2*o+1);
            }
            t[o].dat=t[2*o].dat+t[2*o+1].dat;
        }
    }
```

push_down：(向下更新一层 add 数组)

```
    void push_down(long long o,long l,long mid,long r)
    {
        if (add[o]!=0)
        {
            long rl=2*o,rr=2*o+1;
            __int64 tm=add[o];
            add[rl]+=tm;
            add[rr]+=tm;
            t[rl].dat+=tm*(mid-l+1);
            t[rr].dat+=tm*(r-mid);
            add[o]=0;
        }
    }
```

求和：cin>>ml>>mr>>md; query_sum(ml,mr,1); ->求和：a[ml...mr]

```
    __int64 query_sum(long l,long r,long long o)
    {
        if (o>num) return 0;
        long tl=t[o].l,tr=t[o].r;
        if ((tl==l)&&(tr==r))
            return t[o].dat;
        else
        {
```

```
        long mid=(tl+tr)/2;
        push_down(o,tl,mid,tr);
        if (r<=mid)
            return query_sum(l,r,2*o);
        else if (l>mid)
            return query_sum(l,r,2*o+1);
        else
            return (query_sum(l,mid,2*o)+query_sum(mid+1,r,2*o+1));
    }
}
```

加离散化：(POJ 2528 为例)

我们要重新给压缩后的线段标记起点和终点。

按照通用的离散化方法。。。。

首先依次读入线段端点坐标，存于 post[MAXN][2]中，post[i][0]存第一条线段的起点，post[i][1]存第一条线段的终点，然后用一个结构题数组 line[MAXN]记录信息，line[i].li 记录端点坐标，line[i].num 记录这个点属于哪条线段（可以用正负数表示，负数表示起点，正数表示终点）。假如有 N 条线段，就有 2*N 个端点。然后将 line 数组排序，按照端点的坐标，从小到大排。接着要把线段赋予新的端点坐标了。从左到右按照递增的次序，依次更新端点，假如 2*N 个点中，共有 M 个不同坐标的点，那么线段树的范围就是[1,M]。

```
    memset(v,false,sizeof(v));
    memset(t,0,sizeof(t));
    num=0;

    cin>>n;
    for (int i=1;i<=n;i++)
    {
        cin>>ml>>mr;
        sm[2*i-1].nm=ml;
        sm[2*i].nm=mr;
        sm[2*i-1].ky=i;     //start: >0
        sm[2*i].ky=-i;      //end:   <0
    }
    isort(1,2*n);    //sort: based on sm[i].nm

    tmp=0;
    for (int i=1;i<=2*n;i++)
    {
        if (sm[i].nm!=sm[i-1].nm) tmp++;
        if ((sm[i].nm-sm[i-1].nm==2)&&(i>1)) tmp++;
        int tkey=sm[i].ky;
        if (tkey>0)
            sgm[tkey].st=tmp;
        else if (tkey<0)
            sgm[-tkey].ed=tmp;
    }
```

```
        build(1,tmp,1);

        for (int i=1;i<=n;i++)
        {
            ml=sgm[i].st;   mr=sgm[i].ed;   md=i;
            update(ml,mr,1);        //a[ml..mr]=md;
        }

        ans=0;
        sum(1);
        cout<<ans<<endl;
```

AHOI 2009 seq:

mul[]:乘法用，初值为 1

```
void push_down(long long o,int l,int mid,int r)
{
    t[2*o].dat=(t[2*o].dat*mul[o]+(mid-l+1)*add[o])%p;
    t[2*o+1].dat=(t[2*o+1].dat*mul[o]+(r-mid)*add[o])%p;

    mul[2*o]=(mul[2*o]*mul[o])%p;
    mul[2*o+1]=(mul[2*o+1]*mul[o])%p;

    add[2*o]=(add[2*o]*mul[o]+add[o])%p;
    add[2*o+1]=(add[2*o+1]*mul[o]+add[o])%p;

    mul[o]=1;    add[o]=0;
}
```