求欧拉回路或欧拉路,邻接阵形式,复杂度

```
//求欧拉回路或欧拉路,邻接阵形式,复杂度 O(n^2)
//返回路径长度,path 返回路径(有向图时得到的是反向路径)
//传入图的大小 n 和邻接阵 mat,不相邻点边权 0
//可以有自环与重边,分为无向图和有向图

#define MAXN 100

void find_path_u(int n,int mat[][MAXN],int now,int& step,int* path){
	int i;
	for (i=n-1;i>=0;i--)
		while (mat[now][i]){
			mat[now][i]--,mat[i][now]--;
			find_path_u(n,mat,i,step,path);
		}
	path[step++]=now;
}

void find_path_d(int n,int mat[][MAXN],int now,int& step,int* path){
	int i;
	for (i=n-1;i>=0;i--)
		while (mat[now][i]){
			mat[now][i]--;
			find_path_d(n,mat,i,step,path);
		}
	path[step++]=now;
}
```

```
int euclid_path(int n,int mat[][MAXN],int start,int* path){
	int ret=0;
	find_path_u(n,mat,start,ret,path);
//	find_path_d(n,mat,start,ret,path);
	return ret;
}
```

大数(整数类封装)

```
#include <iostream.h>
#include <string.h>

#define DIGIT	4
#define DEPTH		10000
#define MAX		100
typedef int bignum_t[MAX+1];

int read(bignum_t a,istream& is=cin){
	char buf[MAX*DIGIT+1],ch;
	int i,j;
	memset((void*)a,0,sizeof(bignum_t));
	if (!(is>>buf))	return 0;
	for (a[0]=strlen(buf),i=a[0]/2-1;i>=0;i--)
		ch=buf[i],buf[i]=buf[a[0]-1-i],buf[a[0]-1-i]=ch;
	for (a[0]=(a[0]+DIGIT-1)/DIGIT,j=strlen(buf);j<a[0]*DIGIT;buf[j++]='0');
	for (i=1;i<=a[0];i++)
		for (a[i]=0,j=0;j<DIGIT;j++)
			a[i]=a[i]*10+buf[i*DIGIT-1-j]-'0';
	for (;!a[a[0]]&&a[0]>1;a[0]--);
	return 1;
}

void write(const bignum_t a,ostream& os=cout){
	int i,j;
	for (os<<a[i=a[0]],i--;i;i--)
		for (j=DEPTH/10;j;j/=10)
			os<<a[i]/j%10;
}

int comp(const bignum_t a,const bignum_t b){
	int i;
	if (a[0]!=b[0])
```

```
                return a[0]-b[0];
        for (i=a[0];i;i--)
                if (a[i]!=b[i])
                        return a[i]-b[i];
        return 0;
}

int comp(const bignum_t a,const int b){
        int c[12]={1};
        for (c[1]=b;c[c[0]]>=DEPTH;c[c[0]+1]=c[c[0]]/DEPTH,c[c[0]]%=DEPTH,c[0]++);
        return comp(a,c);
}

int comp(const bignum_t a,const int c,const int d,const bignum_t b){
        int i,t=0,O=-DEPTH*2;
        if (b[0]-a[0]<d&&c)
                return 1;
        for (i=b[0];i>d;i--){
                t=t*DEPTH+a[i-d]*c-b[i];
                if (t>0) return 1;
                if (t<O) return 0;
        }
        for (i=d;i;i--){
                t=t*DEPTH-b[i];
                if (t>0) return 1;
                if (t<O) return 0;
        }
        return t>0;
}

void add(bignum_t a,const bignum_t b){
        int i;
        for (i=1;i<=b[0];i++)
                if ((a[i]+=b[i])>=DEPTH)
                        a[i]-=DEPTH,a[i+1]++;
        if (b[0]>=a[0])
                a[0]=b[0];
        else
                for (;a[i]>=DEPTH&&i<a[0];a[i]-=DEPTH,i++,a[i]++);
        a[0]+=(a[a[0]+1]>0);
}

void add(bignum_t a,const int b){
        int i=1;
        for (a[1]+=b;a[i]>=DEPTH&&i<a[0];a[i+1]+=a[i]/DEPTH,a[i]%=DEPTH,i++);
        for (;a[a[0]]>=DEPTH;a[a[0]+1]=a[a[0]]/DEPTH,a[a[0]]%=DEPTH,a[0]++);
}

void sub(bignum_t a,const bignum_t b){
        int i;
        for (i=1;i<=b[0];i++)
                if ((a[i]-=b[i])<0)
                        a[i+1]--,a[i]+=DEPTH;
        for (;a[i]<0;a[i]+=DEPTH,i++,a[i]--);
        for (;!a[a[0]]&&a[0]>1;a[0]--);
}

void sub(bignum_t a,const int b){
        int i=1;
        for (a[1]-=b;a[i]<0;a[i+1]+=(a[i]-DEPTH+1)/DEPTH,a[i]-=(a[i]-DEPTH+1)/DEPTH*DEPTH,i++);
        for (;!a[a[0]]&&a[0]>1;a[0]--);
}

void sub(bignum_t a,const bignum_t b,const int c,const int d){
        int i,O=b[0]+d;
        for (i=1+d;i<=O;i++)
                if ((a[i]-=b[i-d]*c)<0)
                        a[i+1]+=(a[i]-DEPTH+1)/DEPTH,a[i]-=(a[i]-DEPTH+1)/DEPTH*DEPTH;
        for (;a[i]<0;a[i+1]+=(a[i]-DEPTH+1)/DEPTH,a[i]-=(a[i]-DEPTH+1)/DEPTH*DEPTH,i++);
        for (;!a[a[0]]&&a[0]>1;a[0]--);
}

void mul(bignum_t c,const bignum_t a,const bignum_t b){
        int i,j;
        memset((void*)c,0,sizeof(bignum_t));
        for (c[0]=a[0]+b[0]-1,i=1;i<=a[0];i++)
                for (j=1;j<=b[0];j++)
                        if ((c[i+j-1]+=a[i]*b[j])>=DEPTH)
                                c[i+j]+=c[i+j-1]/DEPTH,c[i+j-1]%=DEPTH;
        for (c[0]+=(c[c[0]+1]>0);!c[c[0]]&&c[0]>1;c[0]--);
}

void mul(bignum_t a,const int b){
        int i;
        for (a[1]*=b,i=2;i<=a[0];i++){
                a[i]*=b;
                if (a[i-1]>=DEPTH)
                        a[i]+=a[i-1]/DEPTH,a[i-1]%=DEPTH;
        }
        for (;a[a[0]]>=DEPTH;a[a[0]+1]=a[a[0]]/DEPTH,a[a[0]]%=DEPTH,a[0]++);
        for (;!a[a[0]]&&a[0]>1;a[0]--);
}

void mul(bignum_t b,const bignum_t a,const int c,const int d){
        int i;
        memset((void*)b,0,sizeof(bignum_t));
        for (b[0]=a[0]+d,i=d+1;i<=b[0];i++)
                if ((b[i]+=a[i-d]*c)>=DEPTH)
                        b[i+1]+=b[i]/DEPTH,b[i]%=DEPTH;
        for (;b[b[0]+1];b[0]++,b[b[0]+1]=b[b[0]]/DEPTH,b[b[0]]%=DEPTH);
        for (;!b[b[0]]&&b[0]>1;b[0]--);
}

void div(bignum_t c,bignum_t a,const bignum_t b){
        int h,l,m,i;
        memset((void*)c,0,sizeof(bignum_t));
        c[0]=(b[0]<a[0]+1)?(a[0]-b[0]+2):1;
        for (i=c[0];i;sub(a,b,c[i]=m,i-1),i--)
                for (h=DEPTH-1,l=0,m=(h+l+1)>>1;h>l;m=(h+l+1)>>1)
                        if (comp(b,m,i-1,a)) h=m-1;
                        else l=m;
        for (;!c[c[0]]&&c[0]>1;c[0]--);
        c[0]=c[0]>1?c[0]:1;
}

void div(bignum_t a,const int b,int& c){
        int i;
        for (c=0,i=a[0];i;c=c*DEPTH+a[i],a[i]=c/b,c%=b,i--);
        for (;!a[a[0]]&&a[0]>1;a[0]--);
}

void sqrt(bignum_t b,bignum_t a){
        int h,l,m,i;
        memset((void*)b,0,sizeof(bignum_t));
        for (i=b[0]=(a[0]+1)>>1;i;sub(a,b,m,i-1),b[i]+=m,i--)
                for (h=DEPTH-1,l=0,b[i]=m=(h+l+1)>>1;h>l;b[i]=m=(h+l+1)>>1)
                        if (comp(b,m,i-1,a)) h=m-1;
                        else l=m;
        for (;!b[b[0]]&&b[0]>1;b[0]--);
        for (i=1;i<=b[0];b[i++]>>=1);
}
```

```
int length(const bignum_t a){
        int t,ret;
        for (ret=(a[0]-1)*DIGIT,t=a[a[0]];t;t/=10,ret++);
        return ret>0?ret:1;
}

int digit(const bignum_t a,const int b){
        int i,ret;
        for (ret=a[(b-1)/DIGIT+1],i=(b-1)%DIGIT;i;ret/=10,i--);
        return ret%10;
}

int zeronum(const bignum_t a){
        int ret,t;
        for (ret=0;!a[ret+1];ret++);
        for (t=a[ret+1],ret*=DIGIT;!(t%10);t/=10,ret++);
        return ret;
}

void comp(int* a,const int l,const int h,const int d){
        int i,j,t;
        for (i=l;i<=h;i++)
                for (t=i,j=2;t>1;j++)
                        while (!(t%j))
                                a[j]+=d,t/=j;
}

void convert(int* a,const int h,bignum_t b){
        int i,j,t=1;
        memset(b,0,sizeof(bignum_t));
        for (b[0]=b[1]=1,i=2;i<=h;i++)
                if (a[i])
                        for (j=a[i];j;t*=i,j--)
                                if (t*i>DEPTH)
                                        mul(b,t),t=1;
        mul(b,t);
}

void combination(bignum_t a,int m,int n){
        int* t=new int[m+1];
        memset((void*)t,0,sizeof(int)*(m+1));
        comp(t,n+1,m,1);
        comp(t,2,m-n,-1);
        convert(t,m,a);
```

```
	delete []t;
}

void permutation(bignum_t a,int m,int n){
	int i,t=1;
	memset(a,0,sizeof(bignum_t));
	a[0]=a[1]=1;
	for (i=m-n+1;i<=m;t*=i++)
		if (t*i>DEPTH)
			mul(a,t),t=1;
	mul(a,t);
}

#define SGN(x) ((x)>0?1:((x)<0?-1:0))
#define ABS(x) ((x)>0?(x):-(x))

int read(bignum_t a,int &sgn,istream& is=cin){
	char str[MAX*DIGIT+2],ch,*buf;
	int i,j;
	memset((void*)a,0,sizeof(bignum_t));
	if (!(is>>str)) return 0;
	buf=str,sgn=1;
	if (*buf=='-') sgn=-1,buf++;
	for (a[0]=strlen(buf),i=a[0]/2-1;i>=0;i--)
		ch=buf[i],buf[i]=buf[a[0]-1-i],buf[a[0]-1-i]=ch;
	for (a[0]=(a[0]+DIGIT-1)/DIGIT,j=strlen(buf);j<a[0]*DIGIT;buf[j++]='0');
	for (i=1;i<=a[0];i++)
		for (a[i]=0,j=0;j<DIGIT;j++)
			a[i]=a[i]*10+buf[i*DIGIT-1-j]-'0';
	for (;!a[a[0]]&&a[0]>1;a[0]--);
	if (a[0]==1&&!a[1]) sgn=0;
	return 1;
}

struct bignum{
	bignum_t num;
	int sgn;
public:
inline bignum(){memset(num,0,sizeof(bignum_t));num[0]=1;sgn=0;}
inline int operator!(){return num[0]==1&&!num[1];}
inline                    bignum&                    operator=(const                    bignum&
a){memcpy(num,a.num,sizeof(bignum_t));sgn=a.sgn;return *this;}
inline                    bignum&                    operator=(const                    int
a){memset(num,0,sizeof(bignum_t));num[0]=1;sgn=SGN(a);add(num,sgn*a);return *this;};
```

```
inline bignum& operator+=(const bignum& a){if(sgn==a.sgn)add(num,a.num);else
if(sgn&&a.sgn){int ret=comp(num,a.num);if(ret>0)sub(num,a.num);else if(ret<0){bignum_t t;
    memcpy(t,num,sizeof(bignum_t));memcpy(num,a.num,sizeof(bignum_t));sub(num,t);sgn=a.
sgn;}else memset(num,0,sizeof(bignum_t)),num[0]=1,sgn=0;}else
if(!sgn)memcpy(num,a.num,sizeof(bignum_t)),sgn=a.sgn;return *this;}
inline bignum& operator+=(const int a){if(sgn*a>0)add(num,ABS(a));else if(sgn&&a){int
ret=comp(num,ABS(a));if(ret>0)sub(num,ABS(a));else if(ret<0){bignum_t t;
    memcpy(t,num,sizeof(bignum_t));memset(num,0,sizeof(bignum_t));num[0]=1;add(num,ABS
(a));sgn=-sgn;sub(num,t);}else memset(num,0,sizeof(bignum_t)),num[0]=1,sgn=0;}else
if(!sgn)sgn=SGN(a),add(num,ABS(a));return *this;}
inline bignum operator+(const bignum& a){bignum
ret;memcpy(ret.num,num,sizeof(bignum_t));ret.sgn=sgn;ret+=a;return ret;}
inline bignum operator+(const int a){bignum
ret;memcpy(ret.num,num,sizeof(bignum_t));ret.sgn=sgn;ret+=a;return ret;}
inline bignum& operator-=(const bignum& a){if(sgn*a.sgn<0)add(num,a.num);else
if(sgn&&a.sgn){int ret=comp(num,a.num);if(ret>0)sub(num,a.num);else if(ret<0){bignum_t t;
    memcpy(t,num,sizeof(bignum_t));memcpy(num,a.num,sizeof(bignum_t));sub(num,t);sgn=-s
gn;}else memset(num,0,sizeof(bignum_t)),num[0]=1,sgn=0;}else
if(!sgn)add(num,a.num),sgn=-a.sgn;return *this;}
inline bignum& operator-=(const int a){if(sgn*a<0)add(num,ABS(a));else if(sgn&&a){int
ret=comp(num,ABS(a));if(ret>0)sub(num,ABS(a));else if(ret<0){bignum_t t;
    memcpy(t,num,sizeof(bignum_t));memset(num,0,sizeof(bignum_t));num[0]=1;add(num,ABS
(a));sub(num,t);sgn=-sgn;}else memset(num,0,sizeof(bignum_t)),num[0]=1,sgn=0;}else
if(!sgn)sgn=-SGN(a),add(num,ABS(a));return *this;}
inline bignum operator-(const bignum& a){bignum
ret;memcpy(ret.num,num,sizeof(bignum_t));ret.sgn=sgn;ret-=a;return ret;}
inline bignum operator-(const int a){bignum
ret;memcpy(ret.num,num,sizeof(bignum_t));ret.sgn=sgn;ret-=a;return ret;}
inline bignum& operator*=(const bignum& a){bignum_t
t;mul(t,num,a.num);memcpy(num,t,sizeof(bignum_t));sgn*=a.sgn;return *this;}
inline bignum& operator*=(const int a){mul(num,ABS(a));sgn*=SGN(a);return *this;}
inline bignum operator*(const bignum& a){bignum
ret;mul(ret.num,num,a.num);ret.sgn=sgn*a.sgn;return ret;}
inline bignum operator*(const int a){bignum
ret;memcpy(ret.num,num,sizeof(bignum_t));mul(ret.num,ABS(a));ret.sgn=sgn*SGN(a);return
ret;}
inline bignum& operator/=(const bignum& a){bignum_t
t;div(t,num,a.num);memcpy(num,t,sizeof(bignum_t));sgn=(num[0]==1&&!num[1])?0:sgn*a.sgn;r
eturn *this;}
inline bignum& operator/=(const int a){int
t;div(num,ABS(a),t);sgn=(num[0]==1&&!num[1])?0:sgn*SGN(a);return *this;}
inline bignum operator/(const bignum& a){bignum ret;bignum_t
t;memcpy(t,num,sizeof(bignum_t));div(ret.num,t,a.num);ret.sgn=(ret.num[0]==1&&!ret.num[1])?
0:sgn*a.sgn;return ret;}
```

```
inline bignum operator/(const int a){bignum ret;int t;memcpy(ret.num,num,sizeof(bignum_t));div(ret.num,ABS(a),t);ret.sgn=(ret.num[0]==1&&!ret.num[1])?0:sgn*SGN(a);return ret;}
inline bignum& operator%=(const bignum& a){bignum_t t;div(t,num,a.num);if (num[0]==1&&!num[1])sgn=0;return *this;}
inline int operator%=(const int a){int t;div(num,ABS(a),t);memset(num,0,sizeof(bignum_t));num[0]=1;add(num,t);return t;}
inline bignum operator%(const bignum& a){bignum ret;bignum_t t;memcpy(ret.num,num,sizeof(bignum_t));div(t,ret.num,a.num);ret.sgn=(ret.num[0]==1&&!ret.num[1])?0:sgn;return ret;}
inline int operator%(const int a){bignum ret;int t;memcpy(ret.num,num,sizeof(bignum_t));div(ret.num,ABS(a),t);memset(ret.num,0,sizeof(bignum_t));ret.num[0]=1;add(ret.num,t);return t;}
inline bignum& operator++(){*this+=1;return *this;}
inline bignum& operator--(){*this-=1;return *this;};
inline int operator>(const bignum& a){return sgn>0?(a.sgn>0?comp(num,a.num)>0:1):(sgn<0?(a.sgn<0?comp(num,a.num)<0:0):a.sgn<0);}
inline int operator>(const int a){return sgn>0?(a>0?comp(num,a)>0:1):(sgn<0?(a<0?comp(num,-a)<0:0):a<0);}
inline int operator>=(const bignum& a){return sgn>0?(a.sgn>0?comp(num,a.num)>=0:1):(sgn<0?(a.sgn<0?comp(num,a.num)<=0:0):a.sgn<=0);}
inline int operator>=(const int a){return sgn>0?(a>0?comp(num,a)>=0:1):(sgn<0?(a<0?comp(num,-a)<=0:0):a<=0);}
inline int operator<(const bignum& a){return sgn<0?(a.sgn<0?comp(num,a.num)>0:1):(sgn>0?(a.sgn>0?comp(num,a.num)<0:0):a.sgn>0);}
inline int operator<(const int a){return sgn<0?(a<0?comp(num,-a)>0:1):(sgn>0?(a>0?comp(num,a)<0:0):a>0);}
inline int operator<=(const bignum& a){return sgn<0?(a.sgn<0?comp(num,a.num)>=0:1):(sgn>0?(a.sgn>0?comp(num,a.num)<=0:0):a.sgn>=0);}
inline int operator<=(const int a){return sgn<0?(a<0?comp(num,-a)>=0:1):(sgn>0?(a>0?comp(num,a)<=0:0):a>=0);}
inline int operator==(const bignum& a){return (sgn==a.sgn)?!comp(num,a.num):0;}
inline int operator==(const int a){return (sgn*a>=0)?!comp(num,ABS(a)):0;}
inline int operator!=(const bignum& a){return (sgn==a.sgn)?comp(num,a.num):1;}
inline int operator!=(const int a){return (sgn*a>=0)?comp(num,ABS(a)):1;}
inline int operator[](const int a){return digit(num,a);}
friend inline istream& operator>>(istream& is,bignum& a){read(a.num,a.sgn,is);return is;}
friend inline ostream& operator<<(ostream& os,const bignum& a){if(a.sgn<0)os<<'-';write(a.num,os);return os;}
friend inline bignum sqrt(const bignum& a){bignum ret;bignum_t t;memcpy(t,a.num,sizeof(bignum_t));sqrt(ret.num,t);ret.sgn=ret.num[0]!=1||ret.num[1];return ret;}
friend inline bignum sqrt(const bignum& a,bignum& b){bignum ret;memcpy(b.num,a.num,sizeof(bignum_t));sqrt(ret.num,b.num);ret.sgn=ret.num[0]!=1||ret.nu
m[1];b.sgn=b.num[0]!=1||ret.num[1];return ret;}
inline int length(){return ::length(num);}
inline int zeronum(){return ::zeronum(num);}
inline bignum C(const int m,const int n){combination(num,m,n);sgn=1;return *this;}
inline bignum P(const int m,const int n){permutation(num,m,n);sgn=1;return *this;}
};
```

二分堆(binary)

```
//二分堆(binary)
//可插入,获取并删除最小(最大)元素,复杂度均 O(logn)
//可更改元素类型,修改比较符号或换成比较函数
#define MAXN 10000
#define _cp(a,b) ((a)<(b))
typedef int elem_t;

struct heap{
	elem_t h[MAXN];
	int n,p,c;
	void init(){n=0;}
	void ins(elem_t e){
		for (p=++n;p>1&&_cp(e,h[p>>1]);h[p]=h[p>>1],p>>=1);
		h[p]=e;
	}
	int del(elem_t& e){
		if (!n) return 0;
		for
(e=h[p=1],c=2;c<n&&_cp(h[c+=(c<n-1&&_cp(h[c+1],h[c]))],h[n]);h[p]=h[c],p=c,c<<=1);
		h[p]=h[n--];
		return 1;
	}
};
```

多边形

```
#include <stdlib.h>
#include <math.h>
#define MAXN 1000
#define offset 10000
#define eps 1e-8
#define zero(x) (((x)>0?(x):-(x))<eps)
#define _sign(x) ((x)>eps?1:((x)<-eps?2:0))
struct point{double x,y;};
struct line{point a,b;};

double xmult(point p1,point p2,point p0){
	return (p1.x-p0.x)*(p2.y-p0.y)-(p2.x-p0.x)*(p1.y-p0.y);
}

//判定凸多边形,顶点按顺时针或逆时针给出,允许相邻边共线
int is_convex(int n,point* p){
	int i,s[3]={1,1,1};
	for (i=0;i<n&&s[1]|s[2];i++)
		s[_sign(xmult(p[(i+1)%n],p[(i+2)%n],p[i]))]=0;
	return s[1]|s[2];
}

//判定凸多边形,顶点按顺时针或逆时针给出,不允许相邻边共线
int is_convex_v2(int n,point* p){
	int i,s[3]={1,1,1};
	for (i=0;i<n&&s[0]&&s[1]|s[2];i++)
		s[_sign(xmult(p[(i+1)%n],p[(i+2)%n],p[i]))]=0;
	return s[0]&&s[1]|s[2];
}

//判点在凸多边形内或多边形边上,顶点按顺时针或逆时针给出
int inside_convex(point q,int n,point* p){
	int i,s[3]={1,1,1};
	for (i=0;i<n&&s[1]|s[2];i++)
		s[_sign(xmult(p[(i+1)%n],q,p[i]))]=0;
	return s[1]|s[2];
}

//判点在凸多边形内,顶点按顺时针或逆时针给出,在多边形边上返回 0
int inside_convex_v2(point q,int n,point* p){
	int i,s[3]={1,1,1};
	for (i=0;i<n&&s[0]&&s[1]|s[2];i++)
		s[_sign(xmult(p[(i+1)%n],q,p[i]))]=0;
	return s[0]&&s[1]|s[2];
}

//判点在任意多边形内,顶点按顺时针或逆时针给出
//on_edge 表示点在多边形边上时的返回值,offset 为多边形坐标上限
int inside_polygon(point q,int n,point* p,int on_edge=1){
	point q2;
	int i=0,count;
	while (i<n)
```

```
		for (count=i=0,q2.x=rand()+offset,q2.y=rand()+offset;i<n;i++)
			if
(zero(xmult(q,p[i],p[(i+1)%n]))&&(p[i].x-q.x)*(p[(i+1)%n].x-q.x)<eps&&(p[i].y-q.y)*(p[(i+1)%n].y-q.
y)<eps)
				return on_edge;
			else if (zero(xmult(q,q2,p[i])))
				break;
			else                                                                                    if
(xmult(q,p[i],q2)*xmult(q,p[(i+1)%n],q2)<-eps&&xmult(p[i],q,p[(i+1)%n])*xmult(p[i],q2,p[(i+1)%n
])<-eps)
				count++;
	return count&1;
}

inline int opposite_side(point p1,point p2,point l1,point l2){
	return xmult(l1,p1,l2)*xmult(l1,p2,l2)<-eps;
}

inline int dot_online_in(point p,point l1,point l2){
	return zero(xmult(p,l1,l2))&&(l1.x-p.x)*(l2.x-p.x)<eps&&(l1.y-p.y)*(l2.y-p.y)<eps;
}

//判线段在任意多边形内,顶点按顺时针或逆时针给出,与边界相交返回 1
int inside_polygon(point l1,point l2,int n,point* p){
	point t[MAXN],tt;
	int i,j,k=0;
	if (!inside_polygon(l1,n,p)||!inside_polygon(l2,n,p))
		return 0;
	for (i=0;i<n;i++)
		if (opposite_side(l1,l2,p[i],p[(i+1)%n])&&opposite_side(p[i],p[(i+1)%n],l1,l2))
			return 0;
		else if (dot_online_in(l1,p[i],p[(i+1)%n]))
			t[k++]=l1;
		else if (dot_online_in(l2,p[i],p[(i+1)%n]))
			t[k++]=l2;
		else if (dot_online_in(p[i],l1,l2))
			t[k++]=p[i];
	for (i=0;i<k;i++)
		for (j=i+1;j<k;j++){
			tt.x=(t[i].x+t[j].x)/2;
			tt.y=(t[i].y+t[j].y)/2;
			if (!inside_polygon(tt,n,p))
				return 0;
		}
	return 1;
}

point intersection(line u,line v){
	point ret=u.a;
	double t=((u.a.x-v.a.x)*(v.a.y-v.b.y)-(u.a.y-v.a.y)*(v.a.x-v.b.x))
			/((u.a.x-u.b.x)*(v.a.y-v.b.y)-(u.a.y-u.b.y)*(v.a.x-v.b.x));
	ret.x+=(u.b.x-u.a.x)*t;
	ret.y+=(u.b.y-u.a.y)*t;
	return ret;
}

point barycenter(point a,point b,point c){
	line u,v;
	u.a.x=(a.x+b.x)/2;
	u.a.y=(a.y+b.y)/2;
	u.b=c;
	v.a.x=(a.x+c.x)/2;
	v.a.y=(a.y+c.y)/2;
	v.b=b;
	return intersection(u,v);
}

//多边形重心
point barycenter(int n,point* p){
	point ret,t;
	double t1=0,t2;
	int i;
	ret.x=ret.y=0;
	for (i=1;i<n-1;i++)
		if (fabs(t2=xmult(p[0],p[i],p[i+1]))>eps){
			t=barycenter(p[0],p[i],p[i+1]);
			ret.x+=t.x*t2;
			ret.y+=t.y*t2;
			t1+=t2;
		}
	if (fabs(t1)>eps)
		ret.x/=t1,ret.y/=t1;
	return ret;
}
```

多边形切割--可用于半平面交

```
//多边形切割
//可用于半平面交
#define MAXN 100
#define eps 1e-8
#define zero(x) (((x)>0?(x):-(x))<eps)
struct point{double x,y;};

double xmult(point p1,point p2,point p0){
	return (p1.x-p0.x)*(p2.y-p0.y)-(p2.x-p0.x)*(p1.y-p0.y);
}

int same_side(point p1,point p2,point l1,point l2){
	return xmult(l1,p1,l2)*xmult(l1,p2,l2)>eps;
}

point intersection(point u1,point u2,point v1,point v2){
	point ret=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));
	ret.x+=(u2.x-u1.x)*t;
	ret.y+=(u2.y-u1.y)*t;
	return ret;
}

//将多边形沿 l1,l2 确定的直线切割在 side 侧切割,保证 l1,l2,side 不共线
void polygon_cut(int& n,point* p,point l1,point l2,point side){
	point pp[100];
	int m=0,i;
	for (i=0;i<n;i++){
		if (same_side(p[i],side,l1,l2))
			pp[m++]=p[i];
		if
(!same_side(p[i],p[(i+1)%n],l1,l2)&&!(zero(xmult(p[i],l1,l2))&&zero(xmult(p[(i+1)%n],l1,l2))))
			pp[m++]=intersection(p[i],p[(i+1)%n],l1,l2);
	}
	for (n=i=0;i<m;i++)
		if (!i||!zero(pp[i].x-pp[i-1].x)||!zero(pp[i].y-pp[i-1].y))
			p[n++]=pp[i];
	if (zero(p[n-1].x-p[0].x)&&zero(p[n-1].y-p[0].y))
		n--;
	if (n<3)
```

```
        n=0;
}
```

多项式求根(牛顿法)

```
/*  牛顿法解多项式的根
    输入：多项式系数 c[]，多项式度数 n，求在[a,b]间的根
    输出：根
    要求保证[a,b]间有根
*/

double fabs( double x )
{
    return (x<0)? -x : x;
}

double f(int m, double c[], double x)
{
    int i;
    double p = c[m];

    for (i=m; i>0; i--)
        p = p*x + c[i-1];
    return p;
}

int newton(double x0, double *r,
           double c[], double cp[], int n,
           double a, double b, double eps)
{
    int MAX_ITERATION = 1000;
    int i = 1;
    double x1, x2, fp, eps2 = eps/10.0;

    x1 = x0;
    while (i < MAX_ITERATION) {
        x2 = f(n, c, x1);
        fp = f(n-1, cp, x1);
        if ((fabs(fp)<0.000000001) && (fabs(x2)>1.0))
            return 0;
        x2 = x1 - x2/fp;
        if (fabs(x1-x2)<eps2) {
            if (x2<a || x2>b)
```

```
                return 0;
            *r = x2;
            return 1;
        }
        x1 = x2;
        i++;
    }
    return 0;
}

double Polynomial_Root(double c[], int n, double a, double b, double eps)
{
    double *cp;
    int i;
    double root;

    cp = (double *)calloc(n, sizeof(double));
    for (i=n-1; i>=0; i--) {
        cp[i] = (i+1)*c[i+1];
    }
    if (a>b) {
            root = a; a = b; b = root;
    }
    if ((!newton(a, &root, c, cp, n, a, b, eps)) &&
        (!newton(b, &root, c, cp, n, a, b, eps)))
            newton((a+b)*0.5, &root, c, cp, n, a, b, eps);
    free(cp);
    if (fabs(root)<eps)
        return fabs(root);
    else
        return root;
}
```

分数

```
struct frac{
    int num,den;
};

double fabs(double x){
    return x>0?x:-x;
}
```

```
int gcd(int a,int b){
        int t;
        if (a<0)
                a=-a;
        if (b<0)
                b=-b;
        if (!b)
                return a;
        while (t=a%b)
                a=b,b=t;
        return b;
}

void simplify(frac& f){
        int t;
        if (t=gcd(f.num,f.den))
                f.num/=t,f.den/=t;
        else
                f.den=1;
}

frac f(int n,int d,int s=1){
        frac ret;
        if (d<0)
                ret.num=-n,ret.den=-d;
        else
                ret.num=n,ret.den=d;
        if (s)
                simplify(ret);
        return ret;
}

frac convert(double x){
        frac ret;
        for (ret.den=1;fabs(x-int(x))>1e-10;ret.den*=10,x*=10);
        ret.num=(int)x;
        simplify(ret);
        return ret;
}

int fraqcmp(frac a,frac b){
        int g1=gcd(a.den,b.den),g2=gcd(a.num,b.num);
        if (!g1||!g2)
                return 0;
```

```
	return b.den/g1*(a.num/g2)-a.den/g1*(b.num/g2);
}

frac add(frac a,frac b){
	int g1=gcd(a.den,b.den),g2,t;
	if (!g1)
		return f(1,0,0);
	t=b.den/g1*a.num+a.den/g1*b.num;
	g2=gcd(g1,t);
	return f(t/g2,a.den/g1*(b.den/g2),0);
}

frac sub(frac a,frac b){
	return add(a,f(-b.num,b.den,0));
}

frac mul(frac a,frac b){
	int t1=gcd(a.den,b.num),t2=gcd(a.num,b.den);
	if (!t1||!t2)
		return f(1,1,0);
	return f(a.num/t2*(b.num/t1),a.den/t1*(b.den/t2),0);
}

frac div(frac a,frac b){
	return mul(a,f(b.den,b.num,0));
}
```

浮点几何函数库

```
//浮点几何函数库
#include <math.h>
#define eps 1e-8
#define zero(x) (((x)>0?(x):-(x))<eps)
struct point{double x,y;};
struct line{point a,b;};

//计算 cross product (P1-P0)x(P2-P0)
double xmult(point p1,point p2,point p0){
	return (p1.x-p0.x)*(p2.y-p0.y)-(p2.x-p0.x)*(p1.y-p0.y);
}
double xmult(double x1,double y1,double x2,double y2,double x0,double y0){
	return (x1-x0)*(y2-y0)-(x2-x0)*(y1-y0);
}
```

```
//计算 dot product (P1-P0).(P2-P0)
double dmult(point p1,point p2,point p0){
	return (p1.x-p0.x)*(p2.x-p0.x)+(p1.y-p0.y)*(p2.y-p0.y);
}
double dmult(double x1,double y1,double x2,double y2,double x0,double y0){
	return (x1-x0)*(x2-x0)+(y1-y0)*(y2-y0);
}

//两点距离
double distance(point p1,point p2){
	return sqrt((p1.x-p2.x)*(p1.x-p2.x)+(p1.y-p2.y)*(p1.y-p2.y));
}
double distance(double x1,double y1,double x2,double y2){
	return sqrt((x1-x2)*(x1-x2)+(y1-y2)*(y1-y2));
}

//判三点共线
int dots_inline(point p1,point p2,point p3){
	return zero(xmult(p1,p2,p3));
}
int dots_inline(double x1,double y1,double x2,double y2,double x3,double y3){
	return zero(xmult(x1,y1,x2,y2,x3,y3));
}

//判点是否在线段上,包括端点
int dot_online_in(point p,line l){
	return zero(xmult(p,l.a,l.b))&&(l.a.x-p.x)*(l.b.x-p.x)<eps&&(l.a.y-p.y)*(l.b.y-p.y)<eps;
}
int dot_online_in(point p,point l1,point l2){
	return zero(xmult(p,l1,l2))&&(l1.x-p.x)*(l2.x-p.x)<eps&&(l1.y-p.y)*(l2.y-p.y)<eps;
}
int dot_online_in(double x,double y,double x1,double y1,double x2,double y2){
	return zero(xmult(x,y,x1,y1,x2,y2))&&(x1-x)*(x2-x)<eps&&(y1-y)*(y2-y)<eps;
}

//判点是否在线段上,不包括端点
int dot_online_ex(point p,line l){
	return
dot_online_in(p,l)&&(!zero(p.x-l.a.x)||!zero(p.y-l.a.y))&&(!zero(p.x-l.b.x)||!zero(p.y-l.b.y));
}
int dot_online_ex(point p,point l1,point l2){
	return
dot_online_in(p,l1,l2)&&(!zero(p.x-l1.x)||!zero(p.y-l1.y))&&(!zero(p.x-l2.x)||!zero(p.y-l2.y));
```

```
}
int dot_online_ex(double x,double y,double x1,double y1,double x2,double y2){
        return
dot_online_in(x,y,x1,y1,x2,y2)&&(!zero(x-x1)||!zero(y-y1))&&(!zero(x-x2)||!zero(y-y2));
}

//判两点在线段同侧,点在线段上返回 0
int same_side(point p1,point p2,line l){
        return xmult(l.a,p1,l.b)*xmult(l.a,p2,l.b)>eps;
}
int same_side(point p1,point p2,point l1,point l2){
        return xmult(l1,p1,l2)*xmult(l1,p2,l2)>eps;
}

//判两点在线段异侧,点在线段上返回 0
int opposite_side(point p1,point p2,line l){
        return xmult(l.a,p1,l.b)*xmult(l.a,p2,l.b)<-eps;
}
int opposite_side(point p1,point p2,point l1,point l2){
        return xmult(l1,p1,l2)*xmult(l1,p2,l2)<-eps;
}

// 点关于直线的对称点 // by lyt
// 缺点：用了斜率
// 也可以利用"点到直线上的最近点"来做，避免使用斜率。
point symmetric_point(point p1, point l1, point l2) {
    point ret;
    if (l1.x > l2.x - eps && l1.x < l2.x + eps) {
        ret.x = (2 * l1.x - p1.x);
        ret.y = p1.y;
    } else {
        double k = (l1.y - l2.y ) / (l1.x - l2.x);
        ret.x = (2*k*k*l1.x + 2*k*p1.y - 2*k*l1.y - k*k*p1.x + p1.x) / (1 + k*k);
        ret.y = p1.y - (ret.x - p1.x ) / k;
    }
    return ret;
}

//判两直线平行
int parallel(line u,line v){
        return zero((u.a.x-u.b.x)*(v.a.y-v.b.y)-(v.a.x-v.b.x)*(u.a.y-u.b.y));
}
int parallel(point u1,point u2,point v1,point v2){
        return zero((u1.x-u2.x)*(v1.y-v2.y)-(v1.x-v2.x)*(u1.y-u2.y));
```

```
}

//判两直线垂直
int perpendicular(line u,line v){
	return zero((u.a.x-u.b.x)*(v.a.x-v.b.x)+(u.a.y-u.b.y)*(v.a.y-v.b.y));
}
int perpendicular(point u1,point u2,point v1,point v2){
	return zero((u1.x-u2.x)*(v1.x-v2.x)+(u1.y-u2.y)*(v1.y-v2.y));
}

//判两线段相交,包括端点和部分重合
int intersect_in(line u,line v){
	if (!dots_inline(u.a,u.b,v.a)||!dots_inline(u.a,u.b,v.b))
		return !same_side(u.a,u.b,v)&&!same_side(v.a,v.b,u);
	return
dot_online_in(u.a,v)||dot_online_in(u.b,v)||dot_online_in(v.a,u)||dot_online_in(v.b,u);
}
int intersect_in(point u1,point u2,point v1,point v2){
	if (!dots_inline(u1,u2,v1)||!dots_inline(u1,u2,v2))
		return !same_side(u1,u2,v1,v2)&&!same_side(v1,v2,u1,u2);
	return
dot_online_in(u1,v1,v2)||dot_online_in(u2,v1,v2)||dot_online_in(v1,u1,u2)||dot_online_in(v2,
u1,u2);
}

//判两线段相交,不包括端点和部分重合
int intersect_ex(line u,line v){
	return opposite_side(u.a,u.b,v)&&opposite_side(v.a,v.b,u);
}
int intersect_ex(point u1,point u2,point v1,point v2){
	return opposite_side(u1,u2,v1,v2)&&opposite_side(v1,v2,u1,u2);
}

//计算两直线交点,注意事先判断直线是否平行!
//线段交点请另外判线段相交(同时还是要判断是否平行!)
point intersection(line u,line v){
	point ret=u.a;
	double t=((u.a.x-v.a.x)*(v.a.y-v.b.y)-(u.a.y-v.a.y)*(v.a.x-v.b.x))
			/((u.a.x-u.b.x)*(v.a.y-v.b.y)-(u.a.y-u.b.y)*(v.a.x-v.b.x));
	ret.x+=(u.b.x-u.a.x)*t;
	ret.y+=(u.b.y-u.a.y)*t;
	return ret;
}
point intersection(point u1,point u2,point v1,point v2){
	point ret=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));
	ret.x+=(u2.x-u1.x)*t;
	ret.y+=(u2.y-u1.y)*t;
	return ret;
}

//点到直线上的最近点
point ptoline(point p,line l){
	point t=p;
	t.x+=l.a.y-l.b.y,t.y+=l.b.x-l.a.x;
	return intersection(p,t,l.a,l.b);
}
point ptoline(point p,point l1,point l2){
	point t=p;
	t.x+=l1.y-l2.y,t.y+=l2.x-l1.x;
	return intersection(p,t,l1,l2);
}

//点到直线距离
double disptoline(point p,line l){
	return fabs(xmult(p,l.a,l.b))/distance(l.a,l.b);
}
double disptoline(point p,point l1,point l2){
	return fabs(xmult(p,l1,l2))/distance(l1,l2);
}
double disptoline(double x,double y,double x1,double y1,double x2,double y2){
	return fabs(xmult(x,y,x1,y1,x2,y2))/distance(x1,y1,x2,y2);
}

//点到线段上的最近点
point ptoseg(point p,line l){
	point t=p;
	t.x+=l.a.y-l.b.y,t.y+=l.b.x-l.a.x;
	if (xmult(l.a,t,p)*xmult(l.b,t,p)>eps)
		return distance(p,l.a)<distance(p,l.b)?l.a:l.b;
	return intersection(p,t,l.a,l.b);
}
point ptoseg(point p,point l1,point l2){
	point t=p;
	t.x+=l1.y-l2.y,t.y+=l2.x-l1.x;
	if (xmult(l1,t,p)*xmult(l2,t,p)>eps)
		return distance(p,l1)<distance(p,l2)?l1:l2;
	return intersection(p,t,l1,l2);
}

//点到线段距离
double disptoseg(point p,line l){
	point t=p;
	t.x+=l.a.y-l.b.y,t.y+=l.b.x-l.a.x;
	if (xmult(l.a,t,p)*xmult(l.b,t,p)>eps)
		return distance(p,l.a)<distance(p,l.b)?distance(p,l.a):distance(p,l.b);
	return fabs(xmult(p,l.a,l.b))/distance(l.a,l.b);
}
double disptoseg(point p,point l1,point l2){
	point t=p;
	t.x+=l1.y-l2.y,t.y+=l2.x-l1.x;
	if (xmult(l1,t,p)*xmult(l2,t,p)>eps)
		return distance(p,l1)<distance(p,l2)?distance(p,l1):distance(p,l2);
	return fabs(xmult(p,l1,l2))/distance(l1,l2);
}

//矢量 V 以 P 为顶点逆时针旋转 angle 并放大 scale 倍
point rotate(point v,point p,double angle,double scale){
	point ret=p;
	v.x-=p.x,v.y-=p.y;
	p.x=scale*cos(angle);
	p.y=scale*sin(angle);
	ret.x+=v.x*p.x-v.y*p.y;
	ret.y+=v.x*p.y+v.y*p.x;
	return ret;
}
```

矩形切割

```
//矩形切割
//intersect 函数构造矩形 a 和 b 的交集
//cut 函数将 b 关于 a 进行切割,用切下的矩形作参数调用 dummy 函数
struct rect{
	int x1,x2,y1,y2;
	rect(){}
	rect(int a,int b,int c,int d):x1(a),x2(b),y1(c),y2(d){}
};

inline rect intersect(rect& a,rect& b){
	return
rect(a.x1>b.x1?a.x1:b.x1,a.x2<b.x2?a.x2:b.x2,a.y1>b.y1?a.y1:b.y1,a.y2<b.y2?a.y2:b.y2);
}

void dummy(rect a){
	//dispose this rect
}

int cut(rect& a,rect b){
	rect c=intersect(a,b);
	if (c.x1>=c.x2||c.y1>=c.y2)
		return 0;
	if (b.x1<c.x1)
		dummy(rect(b.x1,c.x1,b.y1,b.y2)),b.x1=c.x1;
	if (b.x2>c.x2)
		dummy(rect(c.x2,b.x2,b.y1,b.y2)),b.x2=c.x2;
	if (b.y1<c.y1)
		dummy(rect(b.x1,b.x2,b.y1,c.y1)),b.y1=c.y1;
	if (b.y2>c.y2)
		dummy(rect(b.x1,b.x2,c.y2,b.y2)),b.y2=c.y2;
	return 1;
}
```

矩阵

```
define MAXN 100

#define fabs(x) ((x)>0?(x):-(x))
#define zero(x) (fabs(x)<1e-10)

struct mat{
	int n,m;
	double data[MAXN][MAXN];
};

int mul(mat& c,const mat& a,const mat& b){
	int i,j,k;
	if (a.m!=b.n)
		return 0;
	c.n=a.n,c.m=b.m;
	for (i=0;i<c.n;i++)
		for (j=0;j<c.m;j++)
			for (c.data[i][j]=k=0;k<a.m;k++)
				c.data[i][j]+=a.data[i][k]*b.data[k][j];
	return 1;
}

int inv(mat& a){
	int i,j,k,is[MAXN],js[MAXN];
	double t;
	if (a.n!=a.m)
		return 0;
	for (k=0;k<a.n;k++){
		for (t=0,i=k;i<a.n;i++)
			for (j=k;j<a.n;j++)
				if (fabs(a.data[i][j])>t)
					t=fabs(a.data[is[k]=i][js[k]=j]);
		if (zero(t))
			return 0;
		if (is[k]!=k)
			for (j=0;j<a.n;j++)
				t=a.data[k][j],a.data[k][j]=a.data[is[k]][j],a.data[is[k]][j]=t;
		if (js[k]!=k)
			for (i=0;i<a.n;i++)
				t=a.data[i][k],a.data[i][k]=a.data[i][js[k]],a.data[i][js[k]]=t;
		a.data[k][k]=1/a.data[k][k];
		for (j=0;j<a.n;j++)
			if (j!=k)
				a.data[k][j]*=a.data[k][k];
		for (i=0;i<a.n;i++)
			if (i!=k)
				for (j=0;j<a.n;j++)
					if (j!=k)
						a.data[i][j]-=a.data[i][k]*a.data[k][j];
		for (i=0;i<a.n;i++)
			if (i!=k)
				a.data[i][k]*=-a.data[k][k];
	}
	for (k=a.n-1;k>=0;k--){
		for (j=0;j<a.n;j++)
			if (js[k]!=k)
				t=a.data[k][j],a.data[k][j]=a.data[js[k]][j],a.data[js[k]][j]=t;
		for (i=0;i<a.n;i++)
			if (is[k]!=k)
				t=a.data[i][k],a.data[i][k]=a.data[i][is[k]],a.data[i][is[k]]=t;
	}
	return 1;
}
```

```
double det(const mat& a){
	int i,j,k,sign=0;
	double b[MAXN][MAXN],ret=1,t;
	if (a.n!=a.m)
		return 0;
	for (i=0;i<a.n;i++)
		for (j=0;j<a.m;j++)
			b[i][j]=a.data[i][j];
	for (i=0;i<a.n;i++){
		if (zero(b[i][i])){
			for (j=i+1;j<a.n;j++)
				if (!zero(b[j][i]))
					break;
			if (j==a.n)
				return 0;
			for (k=i;k<a.n;k++)
				t=b[i][k],b[i][k]=b[j][k],b[j][k]=t;
			sign++;
		}
		ret*=b[i][i];
		for (k=i+1;k<a.n;k++)
			b[i][k]/=b[i][i];
		for (j=i+1;j<a.n;j++)
			for (k=i+1;k<a.n;k++)
				b[j][k]-=b[j][i]*b[i][k];
	}
	if (sign&1)
		ret=-ret;
	return ret;
}
```

将用边表示的树转化为前序表示的树

```
//将用边表示的树转化为前序表示的树
//传入节点数 n 和邻接表 list[],邻接表必须是双向的,会在函数中释放
//pre[]返回前序表,map[]返回前序表中的节点到原来节点的映射
#define MAXN 10000
struct node{
	int to;
	node* next;
};
```

```
void prenode(int n,node* list[],int* pre,int* map,int* v,int now,int last,int& id){
	node* t;
	int p=id++;
	for (v[map[p]=now]=1,pre[p]=last;list[now];){
		t=list[now],list[now]=t->next;
		if (!v[t->to])
			prenode(n,list,pre,map,v,t->to,p,id);
	}
}

void makepre(int n,node* list[],int* pre,int* map){
	int v[MAXN],id=0,i;
	for (i=0;i<n;v[i++]=0);
	prenode(n,list,pre,map,v,0,-1,id);
}
```

三角形

```
#include <math.h>
struct point{double x,y;};
struct line{point a,b;};

double distance(point p1,point p2){
	return sqrt((p1.x-p2.x)*(p1.x-p2.x)+(p1.y-p2.y)*(p1.y-p2.y));
}

point intersection(line u,line v){
	point ret=u.a;
	double t=((u.a.x-v.a.x)*(v.a.y-v.b.y)-(u.a.y-v.a.y)*(v.a.x-v.b.x))
			/((u.a.x-u.b.x)*(v.a.y-v.b.y)-(u.a.y-u.b.y)*(v.a.x-v.b.x));
	ret.x+=(u.b.x-u.a.x)*t;
	ret.y+=(u.b.y-u.a.y)*t;
	return ret;
}

//外心
point circumcenter(point a,point b,point c){
	line u,v;
	u.a.x=(a.x+b.x)/2;
	u.a.y=(a.y+b.y)/2;
	u.b.x=u.a.x-a.y+b.y;
	u.b.y=u.a.y+a.x-b.x;
	v.a.x=(a.x+c.x)/2;
	v.a.y=(a.y+c.y)/2;
	v.b.x=v.a.x-a.y+c.y;
	v.b.y=v.a.y+a.x-c.x;
	return intersection(u,v);
}

//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 intersection(u,v);
}

//垂心
point perpencenter(point a,point b,point c){
	line u,v;
	u.a=c;
	u.b.x=u.a.x-a.y+b.y;
	u.b.y=u.a.y+a.x-b.x;
	v.a=b;
	v.b.x=v.a.x-a.y+c.y;
	v.b.y=v.a.y+a.x-c.x;
	return intersection(u,v);
}

//重心
//到三角形三顶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 barycenter(point a,point b,point c){
	line u,v;
	u.a.x=(a.x+b.x)/2;
	u.a.y=(a.y+b.y)/2;
	u.b=c;
	v.a.x=(a.x+c.x)/2;
	v.a.y=(a.y+c.y)/2;
	v.b=b;
	return intersection(u,v);
}

//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;
}
```

拓扑排序,邻接阵形式,复杂度 O(n^2)

```
//拓扑排序,邻接阵形式,复杂度 O(n^2)
//如果无法完成排序,返回 0,否则返回 1,ret 返回有序点列
//传入图的大小 n 和邻接阵 mat,不相邻点边权 0
#define MAXN 100

int toposort(int n,int mat[][MAXN],int* ret){
	int d[MAXN],i,j,k;
	for (i=0;i<n;i++)
		for (d[i]=j=0;j<n;d[i]+=mat[j++][i]);
	for (k=0;k<n;ret[k++]=i){
		for (i=0;d[i]&&i<n;i++);
		if (i==n)
			return 0;
		for (d[i]=-1,j=0;j<n;j++)
			d[j]-=mat[i][j];
	}
	return 1;
}
```

网格(pick)

```
#define abs(x) ((x)>0?(x):-(x))
struct point{int x,y;};

int gcd(int a,int b){
	return b?gcd(b,a%b):a;
}

//多边形上的网格点个数
int grid_onedge(int n,point* p){
	int i,ret=0;
	for (i=0;i<n;i++)
		ret+=gcd(abs(p[i].x-p[(i+1)%n].x),abs(p[i].y-p[(i+1)%n].y));
	return ret;
}

//多边形内的网格点个数
int grid_inside(int n,point* p){
	int i,ret=0;
	for (i=0;i<n;i++)
		ret+=p[(i+1)%n].y*(p[i].x-p[(i+2)%n].x);
	return (abs(ret)-grid_onedge(n,p))/2+1;
}
```

线性方程组(gauss)

```
#define MAXN 100
#define fabs(x) ((x)>0?(x):-(x))
#define eps 1e-10

//列主元 gauss 消去求解 a[][]x[]=b[]
//返回是否有唯一解,若有解在 b[]中
int gauss_cpivot(int n,double a[][MAXN],double b[]){
	int i,j,k,row;
	double maxp,t;
	for (k=0;k<n;k++){
		for (maxp=0,i=k;i<n;i++)
			if (fabs(a[i][k])>fabs(maxp))
```

```
                maxp=a[row=i][k];
        if (fabs(maxp)<eps)
            return 0;
        if (row!=k){
            for (j=k;j<n;j++)
                t=a[k][j],a[k][j]=a[row][j],a[row][j]=t;
            t=b[k],b[k]=b[row],b[row]=t;
        }
        for (j=k+1;j<n;j++){
            a[k][j]/=maxp;
            for (i=k+1;i<n;i++)
                a[i][j]-=a[i][k]*a[k][j];
        }
        b[k]/=maxp;
        for (i=k+1;i<n;i++)
            b[i]-=b[k]*a[i][k];
    }
    for (i=n-1;i>=0;i--)
        for (j=i+1;j<n;j++)
            b[i]-=a[i][j]*b[j];
    return 1;
}

//全主元 gauss 消去解 a[][]x[]=b[]
//返回是否有唯一解,若有解在 b[]中
int gauss_tpivot(int n,double a[][MAXN],double b[]){
    int i,j,k,row,col,index[MAXN];
    double maxp,t;
    for (i=0;i<n;i++)
        index[i]=i;
    for (k=0;k<n;k++){
        for (maxp=0,i=k;i<n;i++)
            for (j=k;j<n;j++)
                if (fabs(a[i][j])>fabs(maxp))
                    maxp=a[row=i][col=j];
        if (fabs(maxp)<eps)
            return 0;
        if (col!=k){
            for (i=0;i<n;i++)
                t=a[i][col],a[i][col]=a[i][k],a[i][k]=t;
            j=index[col],index[col]=index[k],index[k]=j;
        }
        if (row!=k){
            for (j=k;j<n;j++)
```

```
                    t=a[k][j],a[k][j]=a[row][j],a[row][j]=t;
                t=b[k],b[k]=b[row],b[row]=t;
            }
            for (j=k+1;j<n;j++){
                a[k][j]/=maxp;
                for (i=k+1;i<n;i++)
                    a[i][j]-=a[i][k]*a[k][j];
            }
            b[k]/=maxp;
            for (i=k+1;i<n;i++)
                b[i]-=b[k]*a[i][k];
        }
        for (i=n-1;i>=0;i--)
            for (j=i+1;j<n;j++)
                b[i]-=a[i][j]*b[j];
        for (k=0;k<n;k++)
            a[0][index[k]]=b[k];
        for (k=0;k<n;k++)
            b[k]=a[0][k];
        return 1;
}
```

圆

```
#include <math.h>
#define eps 1e-8
struct point{double x,y;};

double xmult(point p1,point p2,point p0){
        return (p1.x-p0.x)*(p2.y-p0.y)-(p2.x-p0.x)*(p1.y-p0.y);
}

double distance(point p1,point p2){
        return sqrt((p1.x-p2.x)*(p1.x-p2.x)+(p1.y-p2.y)*(p1.y-p2.y));
}

double disptoline(point p,point l1,point l2){
        return fabs(xmult(p,l1,l2))/distance(l1,l2);
}

point intersection(point u1,point u2,point v1,point v2){
        point ret=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));
        ret.x+=(u2.x-u1.x)*t;
        ret.y+=(u2.y-u1.y)*t;
        return ret;
}

//判直线和圆相交,包括相切
int intersect_line_circle(point c,double r,point l1,point l2){
        return disptoline(c,l1,l2)<r+eps;
}

//判线段和圆相交,包括端点和相切
int intersect_seg_circle(point c,double r,point l1,point l2){
        double t1=distance(c,l1)-r,t2=distance(c,l2)-r;
        point t=c;
        if (t1<eps||t2<eps)
                return t1>-eps||t2>-eps;
        t.x+=l1.y-l2.y;
        t.y+=l2.x-l1.x;
        return xmult(l1,c,t)*xmult(l2,c,t)<eps&&disptoline(c,l1,l2)-r<eps;
}

//判圆和圆相交,包括相切
int intersect_circle_circle(point c1,double r1,point c2,double r2){
        return distance(c1,c2)<r1+r2+eps&&distance(c1,c2)>fabs(r1-r2)-eps;
}

//计算圆上到点 p 最近点,如 p 与圆心重合,返回 p 本身
point dot_to_circle(point c,double r,point p){
        point u,v;
        if (distance(p,c)<eps)
                return p;
        u.x=c.x+r*fabs(c.x-p.x)/distance(c,p);
        u.y=c.y+r*fabs(c.y-p.y)/distance(c,p)*((c.x-p.x)*(c.y-p.y)<0?-1:1);
        v.x=c.x-r*fabs(c.x-p.x)/distance(c,p);
        v.y=c.y-r*fabs(c.y-p.y)/distance(c,p)*((c.x-p.x)*(c.y-p.y)<0?-1:1);
        return distance(u,p)<distance(v,p)?u:v;
}

//计算直线与圆的交点,保证直线与圆有交点
//计算线段与圆的交点可用这个函数后判点是否在线段上
void intersection_line_circle(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=intersection(p,c,l1,l2);
	t=sqrt(r*r-distance(p,c)*distance(p,c))/distance(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;
}

//计算圆与圆的交点,保证圆与圆有交点,圆心不重合
void intersection_circle_circle(point c1,double r1,point c2,double r2,point& p1,point& p2){
	point u,v;
	double t;
	t=(1+(r1*r1-r2*r2)/distance(c1,c2)/distance(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;
	intersection_line_circle(c1,r1,u,v,p1,p2);
}
```

质因数分解

```
//分解质因数
//prime_factor()传入 n, 返回不同质因数的个数
//f 存放质因数，nf 存放对应质因数的个数
//先调用 initprime()，其中第二个 initprime()更快

#include<iostream>
#include<cstdio>
#include<cmath>
using namespace std;
#define MAXN 2001000
#define PSIZE 100000
int plist[PSIZE], pcount=0;
int prime(int n){
	int i;
	if ((n!=2&&!(n%2))||(n!=3&&!(n%3))||(n!=5&&!(n%5))||(n!=7&&!(n%7)))
		return 0;
	for (i=0;plist[i]*plist[i]<=n;++i)
		if (!(n%plist[i]))
			return 0;
	return n>1;
}
void initprime(){
	int i;
	for (plist[pcount++]=2,i=3;i<100000;++i)
		if (prime(i))
			plist[pcount++]=i;
}
int prime_factor(int n, int* f, int *nf) {
	int cnt = 0;
	int n2 = sqrt((double)n);
	for(int i = 0; n > 1 && plist[i] <= n2; ++i)
		if (n % plist[i] == 0) {
			for (nf[cnt] = 0; n % plist[i] == 0; ++nf[cnt], n /= plist[i]);
			f[cnt++] = plist[i];
		}
	if (n > 1) nf[cnt] = 1, f[cnt++] = n;
	return cnt;
}

/*

//产生 MAXN 以内的所有素数
//note:2863311530 就是 10101010101010101010101010101010
//给所有 2 的倍数赋初值
#include <cmath>
#include <iostream>
using namespace std;
#define MAXN 100000000
unsigned int plist[6000000],pcount;
unsigned int isprime[(MAXN>>5)+1];
#define setbitzero(a) (isprime[(a)>>5]&=(~(1<<((a)&31))))
#define setbitone(a) (isprime[(a)>>5]|=(1<<((a)&31)))
#define ISPRIME(a) (isprime[(a)>>5]&(1<<((a)&31)))
void initprime(){
	int i,j,m;
	int t=(MAXN>>5)+1;
	for(i=0;i<t;++i)isprime[i]=2863311530;
	plist[0]=2;setbitone(2);setbitzero(1);
	m=(int)sqrt(MAXN);
	for(pcount=1,i=3;i<=m;i+=2)
		if(ISPRIME(i))
			for(plist[pcount++]=i,j=i<<1;j<=MAXN;j+=i)
				setbitzero(j);
```

```
        if(!(i&1))++i;
        for(;i<=MAXN;i+=2)if(ISPRIME(i))plist[pcount++]=i;
}

*/
```

求置换的循环节,polya 原理

```
//求置换的循环节,polya 原理
//perm[0..n-1]为 0..n-1 的一个置换(排列)
//返回置换最小周期,num 返回循环节个数
#define MAXN 1000

int gcd(int a,int b){
        return b?gcd(b,a%b):a;
}

int polya(int* perm,int n,int& num){
        int i,j,p,v[MAXN]={0},ret=1;
        for (num=i=0;i<n;i++)
                if (!v[i]){
                        for (num++,j=0,p=i;!v[p=perm[p]];j++)
                                v[p]=1;
                        ret*=j/gcd(ret,j);
                }
        return ret;
}
```

周期性方程(追赶法)

```
/*    追赶法解周期性方程
        周期性方程定义：| a1 b1 c1 ...                          |      |      | =    x1
                        |        a2 b2 c2 ...                    |      |      | =    x2
                        |             ...                           | * | X | =    ...
                        | cn-1 ...              an-1 bn-1 |      |      | =    xn-1
                        | bn      cn                        an    |      |      | =    xn
        输入：a[],b[],c[],x[]
        输出：求解结果 X 在 x[]中
*/

void run()
{
        c[0] /= b[0]; a[0] /= b[0]; x[0] /= b[0];
```

```
	for (int i = 1; i < N - 1; i ++) {
		double temp = b[i] - a[i] * c[i - 1];
		c[i] /= temp;
		x[i] = (x[i] - a[i] * x[i - 1]) / temp;
		a[i] = -a[i] * a[i - 1] / temp;
	}
	a[N - 2] = -a[N - 2] - c[N - 2];
	for (int i = N - 3; i >= 0; i --) {
		a[i] = -a[i] - c[i] * a[i + 1];
		x[i] -= c[i] * x[i + 1];
	}
	x[N - 1] -= (c[N - 1] * x[0] + a[N - 1] * x[N - 2]);
	x[N - 1] /= (c[N - 1] * a[0] + a[N - 1] * a[N - 2] + b[N - 1]);
	for (int i = N - 2; i >= 0; i --)
		x[i] += a[i] * x[N - 1];
}
```

子段和--求 sum{[0..n-1]}

```
//求 sum{[0..n-1]}
//维护和查询复杂度均为 O(logn)
//用于动态求子段和,数组内容保存在 sum.a[]中
//可以改成其他数据类型
#include <string.h>
#define lowbit(x) ((x)&((x)^((x)-1)))
#define MAXN 10000
typedef int elem_t;

struct sum{
	elem_t a[MAXN],c[MAXN],ret;
	int n;
	void init(int i){memset(a,0,sizeof(a));memset(c,0,sizeof(c));n=i;}
	void update(int i,elem_t v){for (v-=a[i],a[i++]+=v;i<=n;c[i-1]+=v,i+=lowbit(i));}
	elem_t query(int i){for (ret=0;i;ret+=c[i-1],i^=lowbit(i));return ret;}
};
```

子阵和--求 sum{a[0..m-1][0..n-1]}

```
//求 sum{a[0..m-1][0..n-1]}
//维护和查询复杂度均为 O(logm*logn)
//用于动态求子阵和,数组内容保存在 sum.a[][]中
//可以改成其他数据类型
#include <string.h>
```

```
#define lowbit(x) ((x)&((x)^((x)-1)))
#define MAXN 100
typedef int elem_t;

struct sum{
	elem_t a[MAXN][MAXN],c[MAXN][MAXN],ret;
	int m,n,t;
	void init(int i,int j){memset(a,0,sizeof(a));memset(c,0,sizeof(c));m=i,n=j;}
	void update(int i,int j,elem_t v){
		for (v-=a[i][j],a[i++][j++]+=v,t=j;i<=m;i+=lowbit(i))
			for (j=t;j<=n;c[i-1][j-1]+=v,j+=lowbit(j));
	}
	elem_t query(int i,int j){
		for (ret=0,t=j;i;i^=lowbit(i))
			for (j=t;j;ret+=c[i-1][j-1],j^=lowbit(j));
		return ret;
	}
};
```

字典序全排列与序号的转换

```
//字典序全排列与序号的转换
int perm2num(int n,int *p){
	int i,j,ret=0,k=1;
	for (i=n-2;i>=0;k*=n-(i--))
		for (j=i+1;j<n;j++)
			if (p[j]<p[i])
				ret+=k;
	return ret;
}

void num2perm(int n,int *p,int t){
	int i,j;
	for (i=n-1;i>=0;i--)
		p[i]=t%(n-i),t/=n-i;
	for (i=n-1;i;i--)
		for (j=i-1;j>=0;j--)
			if (p[j]<=p[i])
				p[i]++;
}
```

最长公共单调子序列

```
// 最长公共递增子序列， 时间复杂度 O(n^2 * logn)，空间 O(n^2)

/**
 * n 为 a 的大小, m 为 b 的大小
 * 结果在 ans 中
 * "define _cp(a,b) ((a)<(b))"求解最长严格递增序列
 */
#define MAXN 1000
#define _cp(a,b) ((a)<(b))
typedef int elem_t;

elem_t DP[MAXN][MAXN];
int num[MAXN], p[1<<20];

int LIS(int n, elem_t *a, int m, elem_t *b, elem_t *ans){
        int i, j, l, r, k;

        DP[0][0] = 0;
        num[0] = (b[0] == a[0]);
        for(i = 1; i < m; i++) {
                num[i] = (b[i] == a[0]) || num[i-1];
                DP[i][0] = 0;
        }
        for(i = 1; i < n; i++){
                if(b[0] == a[i] && !num[0]) {
                        num[0] = 1;
                        DP[0][0] = i<<10;
                }

                for(j = 1; j < m; j++){
                        for(k=((l=0)+(r=num[j-1]-1))>>1; l<=r; k=(l+r)>>1)
                                if(_cp(a[DP[j-1][k]>>10], a[i]))
                                        l=k+1;
                                else
                                        r=k-1;

                        if(l < num[j-1] && i == (DP[j-1][l]>>10) ){
                                if(l >= num[j]) DP[j][num[j]++] = DP[j-1][l];
                                else DP[j][l] = _cp(a[DP[j][l]>>10],a[i]) ? DP[j][l] : DP[j-1][l];
                        }
                        if(b[j] == a[i]){
```

```
                for(k=((l=0)+(r=num[j]-1))>>1; l<=r; k=(l+r)>>1)
                    if(_cp(a[DP[j][k]>>10], a[i]))
                        l=k+1;
                    else
                        r=k-1;

                DP[j][l] = (i<<10) + j;
                num[j] += (l>=num[j]);
                p[DP[j][l]] = l ? DP[j][l-1] : -1;
            }
        }
    }

    for (k=DP[m-1][i=num[m-1]-1];i>=0;ans[i--]=a[k>>10],k=p[k]);
    return num[m-1];
}
```

最大公约数欧拉函数

```
int gcd(int a,int b){
    return b?gcd(b,a%b):a;
}

inline int lcm(int a,int b){
    return a/gcd(a,b)*b;
}

//求 1..n-1 中与 n 互质的数的个数
int eular(int n){
    int ret=1,i;
    for (i=2;i*i<=n;i++)
        if (n%i==0){
            n/=i,ret*=i-1;
            while (n%i==0)
                n/=i,ret*=i;
        }
    if (n>1)
        ret*=n-1;
    return ret;
}
```

最大流(邻接阵形式)-- 求网络最大流,邻接阵形式

```
//求网络最大流,邻接阵形式
//返回最大流量,flow 返回每条边的流量
//传入网络节点数 n,容量 mat,源点 source,汇点 sink

#define MAXN 100
#define inf 1000000000

int max_flow(int n,int mat[][MAXN],int source,int sink,int flow[][MAXN]){
	int pre[MAXN],que[MAXN],d[MAXN],p,q,t,i,j;
	if (source==sink) return inf;
	for (i=0;i<n;i++)
		for (j=0;j<n;flow[i][j++]=0);
	for (;;){
		for (i=0;i<n;pre[i++]=0);
		pre[t=source]=source+1,d[t]=inf;
		for (p=q=0;p<=q&&!pre[sink];t=que[p++])
			for (i=0;i<n;i++)
				if (!pre[i]&&j=mat[t][i]-flow[t][i])
					pre[que[q++]=i]=t+1,d[i]=d[t]<j?d[t]:j;
				else if (!pre[i]&&j=flow[i][t])
					pre[que[q++]=i]=-t-1,d[i]=d[t]<j?d[t]:j;
		if (!pre[sink]) break;
		for (i=sink;i!=source;)
			if (pre[i]>0)
				flow[pre[i]-1][i]+=d[sink],i=pre[i]-1;
			else
				flow[i][-pre[i]-1]-=d[sink],i=-pre[i]-1;
	}
	for (j=i=0;i<n;j+=flow[source][i++]);
	return j;
}
```

最大团

```
//最大团
//返回最大团大小和一个方案,传入图的大小 n 和邻接阵 mat
//mat[i][j]为布尔量

#define MAXN 60

void clique(int n, int* u, int mat[][MAXN], int size, int& max, int& bb, int* res, int* rr, int* c) {
	int i, j, vn, v[MAXN];
	if (n) {
		if (size + c[u[0]] <= max) return;
		for (i = 0; i < n + size - max && i < n; ++ i) {
			for (j = i + 1, vn = 0; j < n; ++ j)
				if (mat[u[i]][u[j]])
					v[vn ++] = u[j];
			rr[size] = u[i];
			clique(vn, v, mat, size + 1, max, bb, res, rr, c);
			if (bb) return;
		}
	} else if (size > max) {
		max = size;
		for (i = 0; i < size; ++ i)
			res[i] = rr[i];
		bb = 1;
	}
}

int maxclique(int n, int mat[][MAXN], int *ret) {
	int max = 0, bb, c[MAXN], i, j;
	int vn, v[MAXN], rr[MAXN];
	for (c[i = n - 1] = 0; i >= 0; -- i) {
		for (vn = 0, j = i + 1; j < n; ++ j)
			if (mat[i][j])
				v[vn ++] = j;
		bb = 0;
		rr[0] = i;
		clique(vn, v, mat, 1, max, bb, ret, rr, c);
		c[i] = max;
	}
	return max;
}
```

最大子串匹配,复杂度 O(mn)

```
//最大子串匹配,复杂度 O(mn)
//返回最大匹配值,传入两个串和串的长度,重载返回一个最大匹配
//注意做字符串匹配是串末的'\0'没有置!
//可更改元素类型,更换匹配函数和匹配价值函数
#include <string.h>
#define MAXN 100
#define max(a,b) ((a)>(b)?(a):(b))
```

```
#define _match(a,b) ((a)==(b))
#define _value(a,b) 1
typedef char elem_t;

int str_match(int m,elem_t* a,int n,elem_t* b){
	int match[MAXN+1][MAXN+1],i,j;
	memset(match,0,sizeof(match));
	for (i=0;i<m;i++)
		for (j=0;j<n;j++)
			match[i+1][j+1]=max(max(match[i][j+1],match[i+1][j]),
							(match[i][j]+_value(a[i],b[i]))*_match(a[i],b[j]));
	return match[m][n];
}

int str_match(int m,elem_t* a,int n,elem_t* b,elem_t* ret){
	int match[MAXN+1][MAXN+1],last[MAXN+1][MAXN+1],i,j,t;
	memset(match,0,sizeof(match));
	for (i=0;i<m;i++)
		for (j=0;j<n;j++){
			match[i+1][j+1]=(match[i][j+1]>match[i+1][j]?match[i][j+1]:match[i+1][j]);
			last[i+1][j+1]=(match[i][j+1]>match[i+1][j]?3:1);
			if ((t=(match[i][j]+_value(a[i],b[i]))*_match(a[i],b[j]))>match[i+1][j+1])
				match[i+1][j+1]=t,last[i+1][j+1]=2;
		}
	for (;match[i][j];i-=(last[t=i][j]>1),j-=(last[t][j]<3))
		ret[match[i][j]-1]=(last[i][j]<3?a[i-1]:b[j-1]);
	return match[m][n];
}
```

求最大子段和,复杂度 O(n)

```
//求最大子段和,复杂度 O(n)
//传入串长 n 和内容 list[]
//返回最大子段和,重载返回子段位置(maxsum=list[start]+...+list[end])
//可更改元素类型
typedef int elem_t;

elem_t maxsum(int n,elem_t* list){
	elem_t ret,sum=0;
	int i;
	for (ret=list[i=0];i<n;i++)
		sum=(sum>0?sum:0)+list[i],ret=(sum>ret?sum:ret);
	return ret;
}

elem_t maxsum(int n,elem_t* list,int& start,int& end){
	elem_t ret,sum=0;
	int s,i;
	for (ret=list[start=end=s=i=0];i<n;i++,s=(sum>0?s:i))
		if ((sum=(sum>0?sum:0)+list[i])>ret)
			ret=sum,start=s,end=i;
	return ret;
}
```

最大子阵和--求最大子阵和,复杂度 O(n^3)

```
//求最大子阵和,复杂度 O(n^3)
//传入阵的大小 m,n 和内容 mat[][]
//返回最大子阵和,重载返回子阵位置(maxsum=list[s1][s2]+...+list[e1][e2])
//可更改元素类型
#define MAXN 100
typedef int elem_t;

elem_t maxsum(int m,int n,elem_t mat[][MAXN]){
	elem_t matsum[MAXN][MAXN+1],ret,sum;
	int i,j,k;
	for (i=0;i<m;i++)
		for (matsum[i][j=0]=0;j<n;j++)
			matsum[i][j+1]=matsum[i][j]+mat[i][j];
	for (ret=mat[0][j=0];j<n;j++)
		for (k=j;k<n;k++)
			for (sum=0,i=0;i<m;i++)
				sum=(sum>0?sum:0)+matsum[i][k+1]-matsum[i][j],ret=(sum>ret?sum:ret);
	return ret;
}

elem_t maxsum(int m,int n,elem_t mat[][MAXN],int& s1,int& s2,int& e1,int& e2){
	elem_t matsum[MAXN][MAXN+1],ret,sum;
	int i,j,k,s;
	for (i=0;i<m;i++)
		for (matsum[i][j=0]=0;j<n;j++)
			matsum[i][j+1]=matsum[i][j]+mat[i][j];
	for (ret=mat[s1=e1=0][s2=e2=j=0];j<n;j++)
		for (k=j;k<n;k++)
			for (sum=0,s=i=0;i<m;i++,s=(sum>0?s:i))
				if ((sum=(sum>0?sum:0)+matsum[i][k+1]-matsum[i][j])>ret)
```

```
                        ret=sum,s1=s,s2=i,e1=j,e2=k;
    return ret;
}
```

最小顶点割集

```
//最小顶点割集
#define MAXN 100
#define inf 1000000000

int max_flow(int n,int mat[][MAXN],int source,int sink){
    int v[MAXN],c[MAXN],p[MAXN],ret=0,i,j;
    for (;;){
        for (i=0;i<n;i++)
            v[i]=c[i]=0;
        for (c[source]=inf;;){
            for (j=-1,i=0;i<n;i++)
                if (!v[i]&&c[i]&&(j==-1||c[i]>c[j]))
                    j=i;
            if (j<0) return ret;
            if (j==sink) break;
            for (v[j]=1,i=0;i<n;i++)
                if (mat[j][i]>c[i]&&c[j]>c[i])
                    c[i]=mat[j][i]<c[j]?mat[j][i]:c[j],p[i]=j;
        }
        for (ret+=j=c[i=sink];i!=source;i=p[i])
            mat[p[i]][i]-=j,mat[i][p[i]]+=j;
    }
}

int min_vertex_cut(int n,int mat[][MAXN],int source,int sink,int* set){
    int m0[MAXN][MAXN],m[MAXN][MAXN],i,j,k,ret=0,last;
    if (source==sink||mat[source][sink])
        return -1;
    for (i=0;i<n+n;i++)
        for (j=0;j<n+n;j++)
            m0[i][j]=0;
    for (i=0;i<n;i++)
        for (j=0;j<n;j++)
            if (mat[i][j])
                m0[i][n+j]=inf;
    for (i=0;i<n;i++)
        m0[n+i][i]=1;
```

```
	for (i=0;i<n+n;i++)
		for (j=0;j<n+n;j++)
			m[i][j]=m0[i][j];
	last=max_flow(n+n,m,source,n+sink);
	for (k=0;k<n&&last;k++)
		if (k!=source&&k!=sink){
			for (i=0;i<n+n;i++)
				for (j=0;j<n+n;j++)
					m[i][j]=m0[i][j];
			m[n+k][k]=0;
			if (max_flow(n+n,m,source,n+sink)<last){
				set[ret++]=k;
				m0[n+k][k]=0;
				last--;
			}
		}
	return ret;
}
```

最小生成树(prim 邻接阵形式)

```
//无向图最小生成树,prim 算法,邻接阵形式,复杂度 O(n^2)
//返回最小生成树的长度,传入图的大小 n 和邻接阵 mat,不相邻点边权 inf
//可更改边权的类型,pre[]返回树的构造,用父结点表示,根节点(第一个)pre 值为-1
//必须保证图的连通的!
#define MAXN 200
#define inf 1000000000
typedef double elem_t;

elem_t prim(int n,elem_t mat[][MAXN],int* pre){
	elem_t min[MAXN],ret=0;
	int v[MAXN],i,j,k;
	for (i=0;i<n;i++)
		min[i]=inf,v[i]=0,pre[i]=-1;
	for (min[j=0]=0;j<n;j++){
		for (k=-1,i=0;i<n;i++)
			if (!v[i]&&(k==-1||min[i]<min[k]))
				k=i;
		for (v[k]=1,ret+=min[k],i=0;i<n;i++)
			if (!v[i]&&mat[k][i]<min[i])
				min[i]=mat[pre[i]=k][i];
	}
	return ret;
```

}

北大 ACM-题型分类的代码

主流算法：

1.搜索　//回溯

2.DP（动态规划）

3.贪心
4.图论　//Dijkstra、最小生成树、网络流

5.数论　//解模线性方程

6.计算几何　//凸壳、同等安置矩形的并的面积与周长

7.组合数学　//Polya 定理

8.模拟

9.数据结构　//并查集、堆

10.博弈论

1、排序

1423, 1694, 1723, 1727, 1763, 1788, 1828, 1838, 1840, 2201, 2376, 2377, 2380, 1318, 1877,

1928, 1971, 1974, 1990, 2001, 2002, 2092, 2379,

1002（需要字符处理，排序用快排即可） 1007（稳定的排序） 2159（题意较难懂） 2231
2371（简单排

序） 2388（顺序统计算法） 2418（二叉排序树）

2、搜索、回溯、遍历

1022 1111 1118 1129 1190 1562 1564 1573 1655 2184 2225 2243 2312 2362 2378 2386

1010,1011,1018,1020,1054,1062,1256,1321,1363,1501，

1650,1659,1664,1753,2078,2083,2303,2310,2329

简单：1128, 1166, 1176, 1231, 1256, 1270, 1321, 1543, 1606, 1664, 1731, 1742, 1745, 1847,

1915, 1950, 2038, 2157, 2182, 2183, 2381, 2386, 2426,
不易：1024, 1054, 1117, 1167, 1708, 1746, 1775, 1878, 1903, 1966, 2046, 2197, 2349,
推荐：1011, 1190, 1191, 1416, 1579, 1632, 1639, 1659, 1680, 1683, 1691, 1709, 1714, 1753,

1771, 1826, 1855, 1856, 1890, 1924, 1935, 1948, 1979, 1980, 2170, 2288, 2331, 2339,

2340,1979（和迷宫类似） 1980（对剪枝要求较高）

3、历法

1008 2080 （这种题要小心）

4、枚举

1012，1046， 1387， 1411， 2245， 2326， 2363， 2381，1054（剪枝要求较高），1650（小数的精

度问题）

5、数据结构的典型算法

容易：1182, 1656, 2021, 2023, 2051, 2153, 2227, 2236, 2247, 2352, 2395,
不易：1145, 1177, 1195, 1227, 1661, 1834,
推荐：1330, 1338, 1451, 1470, 1634, 1689, 1693, 1703, 1724, 1988, 2004, 2010, 2119, 2274,

1125(弗洛伊德算法) ， 2421（图的最小生成树）

6、 动态规划

1037 A decorative fence、

1050 To the Max、

1088 滑雪、

1125 Stockbroker Grapevine、

1141 Brackets Sequence、

1159 Palindrome、

1160 Post Office、

1163 The Triangle、

1458 Common Subsequence、

1579 Function Run Fun、

1887 Testing the CATCHER、

1953 World Cup Noise、

2386 Lake Counting

7、贪心

1042, 1065, 1230, 1323, 1477, 1716, 1784,1328 1755（或用单纯形方法），2054，1017， 1328，

1862， 1922 ， 2054， 2209， 2313， 2325， 2370。

8、模拟

容易：1006, 1008, 1013, 1016, 1017, 1169, 1298, 1326, 1350, 1363, 1676, 1786, 1791, 1835, 1970, 2317, 2325, 2390,

不易：1012, 1082, 1099, 1114, 1642, 1677, 1684, 1886,1281 1928 2083 2141 2015

9、递归

1664

10、字符串处理

1488, 1598, 1686, 1706, 1747, 1748, 1750, 1760, 1782, 1790, 1866, 1888, 1896, 1951, 2003, 2121, 2141, 2145, 2159, 2337, 2359, 2372, 2406, 2408, 1016 1051 1126 1318 1572 1917 1936 2039 2083 2136 2271 2317 2330， 2121 2403

11、数论

1006,1014,1023,1061,1152,1183,1730,2262

12、几何有关的题目

凸包：1113, 1228, 1794, 2007, 2187,1113 wall， 2187 beauty contest

容易：1319, 1654, 1673, 1675, 1836, 2074, 2137, 2318,
不易：1685, 1687, 1696, 1873, 1901, 2172, 2333,

13、任意精度运算、数字游戏、高精度计算

1001 1023 1047 1060 1079 1131 1140 1142 1207 1220 1284 1289 1306 1316 1338 1405 1454 1503

1504 1519 1565 1650 1969 2000 2006 2081 2247 2262 2305 2316 2389
1001, 1220, 1405, 1503,1001（高精度乘法） 2413(高精度加法，还有二分查找)

14、概率统计

1037,1050

15、小费用最大流、最大流

2195 going home，2400 supervisor, supervisee，1087 a plug for UNIX，1149 PIGS，1273 drainage ditches， 1274 the perfect stall， 1325 machine schedule， 1459 power network， 2239 selecting courses

16、压缩存储的 DP

1038 bugs integrated inc， 1185 炮兵阵地， 2430 lazy cow

17、最长公共子串（LCS）

1080 human gene functions， 1159 palindrome， 1458 common subsequence， 2192 zipper

18、图论及组合数学

2421 Constructing Roads、

2369 Permutations、

2234 Matches Game、

2243 Knight Moves、

2249 Binomial Showdown、

2255 Tree Recovery、

2084 Game of Connections、

1906 Three powers、

1833 排列、

1850 Code、

1562 Oil Deposits、

1496 Word Index、

1306 Combinations、

1125 Stockbroker Grapevine、

1129 Channel Allocation、

1146 ID Codes、

1095 Trees Made to Order、找规律

2247 Humble Numbers、

2309 BST、

2346 Lucky tickets、

2370 Democracy in danger、

2365 Rope、

2101 Honey and Milk Land
2028 When Can We Meet?、

2084 Game of Connections、

1915 Knight Moves、

1922 Ride to School、

1941 The Sierpinski Fractal、

1953 World Cup Noise、

1958 Strange Towers of Hanoi、

1969 Count on Canton、

1806 Manhattan 2025、

1809 Regetni、

1844 Sum、

1870 Bee Breeding、

1702 Eva\'s Balance、

1728 A flea on a chessboard、

1604 Just the Facts、

1642 Stacking Cubes、

1656 Counting Black、

1657 Distance on Chessboard、

1662 CoIns、

1663 Number Steps、

1313 Booklet Printing、

1316 Self Numbers、

1320 Street Numbers、

1323 Game Prediction、

1338 Ugly Numbers、

1244 Slots of Fun、

1250 Tanning Salon、

1102 LC-Display、

1147 Binary codes、

1013 Counterfeit Dollar、

19、博弈类

1067 取石子游戏、

1740 A New Stone Game、

2234 Matches Game、

1082 Calendar Game 、

2348 Euclid\'s Game、

2413 How many Fibs?、

2419 Forest

20、简单、模拟题

1001 Exponentiation 、

1002 487-3279、

1003 Hangover 、

1701 Dissatisfying Lift、

2301 Beat the Spread!、

2304 Combination Lock、

2328 Guessing Game、

2403 Hay Points 、

2406 Power Strings、

2339 Rock, Scissors, Paper、

2350 Above Average、

2218 Does This Make Me Look Fat?、

2260 Error Correction、

2262 Goldbach\'s Conjecture、

2272 Bullseye、

2136 Vertical Histogram、

2174 Decoding Task、

2183 Bovine Math Geniuses、

2000 Gold Coins、

2014 Flow Layout、

2051 Argus、

2081 Calendar、

1918 Ranking List、

1922 Ride to School、

1970 The Game、

1972 Dice Stacking、

1974 The Happy Worm、

1978 Hanafuda Shuffle、

1979 Red and Black、

1617 Crypto Columns、

1666 Candy Sharing Game、

1674 Sorting by Swapping、

1503 Integer Inquiry、

1504 Adding Reversed Numbers、

1528 Perfection、

1546 Basically Speaking、

1547 Clay Bully、

1573 Robot Motion、

1575 Easier Done Than Said?、

1581 A Contesting Decision、

1590 Palindromes、

1454 Factorial Frequencies、

1363 Rails、

1218 THE DRUNK JAILER、

1281 MANAGER、

1132 Border、

1028 Web Navigation、

21、初等数学

1003 Hangover、

1045 Bode Plot、

1254 Hansel and Grethel、

1269 Intersecting Lines、

1401 Factorial、

1410 Intersection、

2363 Blocks　、

2365 Rope、

2242 The Circumference of the Circle、

2291 Rotten Ropes、

2295 A DP Problem、

2126 Factoring a Polynomial、

2191 Mersenne Composite Numbers、

2196 Specialized Four-Digit Numbers、

1914 Cramer\'s Rule、

1835　宇航员、

1799 Yeehaa!、

1607 Deck、

1244 Slots of Fun、

1269 Intersecting Lines、

1299 Polar Explorer、

1183 反正切函数的应用、

22、匹配

1274, 1422, 1469, 1719, 2060, 2239,

经典

1011（搜索好题）
1012（学会打表）
1013
1019（它体现了很多此类问题的特点）
1050（绝对经典的 dp）
1088（dp 好题）
1157（花店，经典的 dp）
1163（怎么经典的 dp 那么多呀？？？）
1328（贪心）
1458（最长公共子序列）
1647（很好的真题，考临场分析准确和下手迅速）
1654（学会多边形面积的三角形求法）
1655（一类无根树的 dp 问题）
1804（逆序对）
2084（经典组合数学问题）
2187（用凸包求最远点对，求出凸包后应该有 O(N)的求法，可我就是调不出来）
2195（二分图的最佳匹配）
2242（计算几何经典）
2295（等式处理）
2353（dp，但要记录最佳路径）
2354（立体解析几何）
2362（搜索好题）
2410（读懂题是关键）
2411（经典 dp）

趣味

1067（很难的数学，但仔细研究，是一片广阔的领域）
1147（有 O(n)的算法，需要思考）
1240（直到一棵树的先序和后序遍历，那么有几种中序遍历呢？dp）
1426（是数论吗？错，是图论！）
1648（别用计算几何，用整点这个特点绕过精度的障碍吧）
1833（找规律）
1844（貌似 dp 或是搜索，其实是道有趣的数学题）
1922（贪心，哈哈）
2231
2305（不需要高精度噢）
2328（要仔细噢）
2356（数论知识）
2359（约瑟夫问题变种）
2392（有趣的问题）

很繁的题

1001
1008
1087（构图很烦，还有二分图的最大匹配）
1128（USACO）
1245
1329
1550（考的是读题和理解能力）
1649（dp）
2200（字符串处理+枚举）
2358（枚举和避免重复都很烦）
2361（仔细仔细再仔细）

难题

1014（数学证明比较难，但有那种想法更重要）
1037（比较难的 dp）
1405（高精度算法也分有等级之分，不断改进吧）
2002（不知道有没有比 O(n^2*logn)更有的算法？）
2054（极难，很强的思考能力）
2085（组合数学）
2414（dp，但要剪枝）
2415（搜索）
2423（计算几何+统计）

多解题

1002（可以用排序，也可以用统计的方法）
1338（搜索和 dp 都可以）
1664（搜索和 dp 都练一练吧）
2082（这可是我讲的题噢）
2352（桶排和二叉树都行）

Note:
1011: 很经典的剪支
1014: 难在数学上
1017: 严格的数学证明貌似不容易
1021: 有点繁,考察对图形进行各种旋转的处理
1083: 巧妙的思考角度
1150: 分奇偶讨论,lg(n)算法
1218: 三行就够了,虽然简单,但也有优劣之别
1505: 二分加贪心
1654: 做法也许很多吧,本人用有向面积做的
1674: 计算圈的个数(算是 graph 吧)
1700: 数学证明不容易
1742: O(m*n)的算法
1863: 要耐心地慢慢写…^_^
1988: 并查集
2051: 堆
2078: 不难，但剪支可以做到很好
2082::O(n),你想到了吗？
2084: 卡特兰数
2182: 线段树
2195: 最小费用最大流
2234: 经典博弈算法
2236: 并查集
2299: 二分思想
2395: Kruskal 最小生成树的拓展
2406: KMP
2411: 用二进制串来表示状态