Source code library for ACM/ICPC in C++

Hao Fu

School of Computer Science and Technology
University of Science and Technology of China

September 21, 2010
Rev. 334

Contents

1 Class

1.1 Big Integer

```
#define iszero(t) (t.len==1&&t.s[0]==0)
#define setlen(l,t) t.len=l;while(t.len>1&&t.s[t.len-1]==0) t.len--
const int maxlen=100;
struct bigint
{
    int len,s[maxlen];
    bigint() {*this=0;}
    bigint(int a) {*this=a;}
    bigint(const char *a) {*this=a;}
    bigint operator=(int);
    bigint operator=(const char*);
    bigint operator=(const bigint&); //Optional
    friend ostream& operator<<(ostream&,const bigint&);
    bigint operator+(const bigint&);
    bigint operator-(const bigint&);
    bigint operator*(const bigint&);
    bigint operator/(const bigint&); //Require - cmp
    bigint operator%(const bigint&); //Require - cmp
    static int cmp(const bigint&,const bigint&);
    static bigint sqrt(const bigint&); //Require - * cmp
};
bigint bigint::operator=(int a)
{
    len=0;
    do{s[len++]=a%10;a/=10;} while(a>0);
    return *this;
}
bigint bigint::operator=(const char *a)
{
    len=strlen(a);
    for(int i=0;i<len;i++) s[i]=a[len-i-1]-'0';
    return *this;
}
bigint bigint::operator=(const bigint &a)
{
    len=a.len;
    memcpy(s,a.s,sizeof(*s)*len);
    return *this;
}
ostream& operator<<(ostream &os,const bigint &a)
{
    for(int i=a.len-1;i>=0;i--) os<<a.s[i];
    return os;
}
bigint bigint::operator+(const bigint &a)
{
    bigint b;
    b.s[b.len=max(len,a.len)]=0;
    for(int i=0;i<b.len;i++) b.s[i]=(i<len?s[i]:0)+(i<a.len?a.s[i]:0);
    for(int i=0;i<b.len;i++)
        if(b.s[i]>=10) {b.s[i]-=10;b.s[i+1]++;}
    if(b.s[b.len]) b.len++;
    return b;
}
//Make sure *this>=a
bigint bigint::operator-(const bigint &a)
{
    bigint b;
    for(int i=0;i<len;i++) b.s[i]=s[i]-(i<a.len?a.s[i]:0);
    for(int i=0;i<len;i++)
        if(b.s[i]<0) {b.s[i]+=10;b.s[i+1]--;}
    setlen(len,b);
    return b;
}
bigint bigint::operator*(const bigint &a)
{
```

```
    bigint b;
    memset(b.s,0,sizeof(*s)*(len+a.len+1));
    for(int i=0;i<len;i++)
        for(int j=0;j<a.len;j++) b.s[i+j]+=s[i]*a.s[j];
    for(int i=0;i<len+a.len;i++) {b.s[i+1]+=b.s[i]/10;b.s[i]%=10;}
    setlen(len+a.len+1,b);
    return b;
}
bigint bigint::operator/(const bigint &a)
{
    bigint b,c;
    for(int i=len-1;i>=0;i--)
    {
        if(!iszero(b))
        {
            for(int j=b.len;j>0;j--) b.s[j]=b.s[j-1];
            b.len++;
        }
        b.s[0]=s[i];c.s[i]=0;
        while(cmp(b,a)>=0) {b=b-a;c.s[i]++;}
    }
    setlen(len,c);
    return c;
}
bigint bigint::operator%(const bigint &a)
{
    bigint b;
    for(int i=len-1;i>=0;i--)
    {
        if(!iszero(b))
        {
            for(int j=b.len;j>0;j--) b.s[j]=b.s[j-1];
            b.len++;
        }
        b.s[0]=s[i];
        while(cmp(b,a)>=0) b=b-a;
    }
    return b;
}
int bigint::cmp(const bigint &a,const bigint &b)
{
    if(a.len<b.len) return -1;
    else if(a.len>b.len) return 1;
    for(int i=a.len-1;i>=0;i--)
        if(a.s[i]!=b.s[i]) return a.s[i]-b.s[i];
    return 0;
}
bigint bigint::sqrt(const bigint &a)
{
    int n=(a.len-1)/2,p;
    bigint b,d;
    b.len=n+1;
    for(int i=n;i>=0;i--)
    {
        if(!iszero(d))
        {
            for(int j=d.len+1;j>1;j--) d.s[j]=d.s[j-2];
            d.s[0]=a.s[i*2];d.s[1]=a.s[i*2+1];
            d.len+=2;
        }
        else d=a.s[i*2]+(i*2+1<a.len?a.s[i*2+1]*10:0);
        bigint c;
        c.s[1]=0;
        for(int j=1;j<=n-i;j++)
        {
            c.s[j]+=b.s[i+j]<<1;
            if(c.s[j]>=10) {c.s[j+1]=1;c.s[j]-=10;} else c.s[j+1]=0;
        }
        c.len=n-i+1+c.s[n-i+1];
        for(p=1;;p++)
        {
            c.s[0]=p;
```

```
            if(cmp(d,c*p)<0) break;
        }
        b.s[i]=c.s[0]=p-1;
        d=d-c*(p-1);
    }
    return b;
}
```

1.2 Fraction

```
int gcd(int a,int b)
{
    while(b) {int t=a%b;a=b;b=t;}
    return a<0?-a:a;
}
int lcm(int a,int b)
{
    return a/gcd(a,b)*b;
}
struct Frac
{
    int a,b;
    Frac() {a=0;b=1;}
    Frac(int x,int y)
    {
        int t=gcd(x,y);
        if(y<0) t=-t;
        a=x/t;b=y/t;
    }
    Frac operator+(Frac z)
    {
        Frac t(a*z.b+b*z.a,b*z.b);
        int d=gcd(t.a,t.b);
        t.a/=d;t.b/=d;
        return t;
    }
    Frac operator-(Frac z)
    {
        Frac t(a*z.b-b*z.a,b*z.b);
        int d=gcd(t.a,t.b);
        t.a/=d;t.b/=d;
        return t;
    }
    Frac operator*(Frac z)
    {
        int x=gcd(z.a,b),y=gcd(z.b,a);
        z.a=z.a/x*(a/y);z.b=z.b/y*(b/x);
        return z;
    }
    Frac operator/(Frac z)
    {
        int x=gcd(a,z.a),y=gcd(b,z.b);
        Frac t(a/x*(z.b/y),b/y*(z.a/x));
        if(z.b<0) {z.a=-z.a;z.b=-z.b;}
        return t;
    }
};
```

1.3 Matrix

```
struct Matrix
{
    int n,a[maxn][maxn];
    Matrix(int size,int dia=0)
    {
        n=size;
        memset(a,0,sizeof(a));
        for(int i=0;i<n;i++) a[i][i]=dia;
    }
    Matrix operator+(const Matrix &x)
    {
        Matrix y(n);
        for(int i=0;i<n;i++)
```

```
            for(int j=0;j<n;j++) y.a[i][j]=a[i][j]+x.a[i][j];
        return y;
    }
    Matrix operator-(const Matrix &x)
    {
        Matrix y(n);
        for(int i=0;i<n;i++)
            for(int j=0;j<n;j++) y.a[i][j]=a[i][j]-x.a[i][j];
        return y;
    }
    Matrix operator*(const Matrix &x)
    {
        Matrix y(n);
        for(int i=0;i<n;i++)
            for(int j=0;j<n;j++)
                for(int k=0;k<n;k++) y.a[i][j]+=a[i][k]*x.a[k][j];
        return y;
    }
    Matrix pow(int p)
    {
        Matrix y(n,1),z=*this;
        for(int i=1;i<=p;i<<=1)
        {
            if(i&p) y=y*z;
            z=z*z;
        }
        return y;
    }
};
```

2 Graph Theory

2.1 Minimum Spanning Tree – Kruskal

```
//Using Union-find
struct edge{int u,v,w;};
bool Kruskal_cmp(const edge *a,const edge *b)
{
    return a->w<b->w;
}
int Kruskal(int n,int m,edge e[],int ret[]) //Return -1 if no tree found.
{
    if(n==1) return 0;
    static edge *d[maxm];
    for(int i=0;i<m;i++) d[i]=e+i;
    sort(d,d+m,Kruskal_cmp);
    int f[maxn],c=0;
    for(int i=0;i<n;i++) MakeSet(f,i);
    for(int i=0,j=0;i<m;i++)
        if(!Same(f,d[i]->u,d[i]->v))
        {
            Union(f,d[i]->u,d[i]->v);
            c+=d[i]->w;
            ret[j]=d[i]-e;
            if(++j==n-1) return c;
        }
    return -1;
}
```

2.2 Minimum Branching for Directed Graph – Edmonds

```
struct edge {int v,w,next;};
//Finds a minimum branching for a directed graph (similar to a minimum spanning tree).
//Check whether the graph is connected by yourself.
//f[]: Reversed graph. v's parent is e[par[v]].v
int Edmonds(int f[],edge e[],int n,int root,int par[])
{
    int ans=0;
    for(int i=0;i<n;i++)
    {
        if(i==root) continue;
```

```
        int w=INT_MAX,d=-1;
        for(int j=f[i];j!=-1;j=e[j].next)
            if(w>e[j].w) {w=e[j].w;d=j;}
        par[i]=d;ans+=w;
    }
    par[root]=-1;
    int m=n,u[maxn*2],v[maxn*2];
    for(int i=0;i<n*2;i++) v[i]=i;
    for(int i=0;i<m;i++)
    {
        if(i==root||i!=v[i]) continue;
        memset(u,255,sizeof(u));
        int p=i;
        while(u[p]==-1&&par[p]!=-1)
        {
            u[p]=i;
            p=v[e[par[p]].v];
        }
        if(u[p]!=i||par[p]==-1) continue;
        while(p!=m)
        {
            v[p]=m;
            p=v[v[e[par[p]].v]];
        }
        int w=INT_MAX,c=-1,d=-1;
        for(int j=0;j<n;j++)
            if(v[v[j]]==m)
            {
                for(int k=f[j];k!=-1;k=e[k].next)
                {
                    if(v[v[e[k].v]]==m) continue;
                    e[k].w-=e[par[v[j]==m?j:v[j]]].w;
                    if(w>e[k].w) {w=e[k].w;c=j;d=k;}
                }
                v[j]=m;
            }
        par[m]=par[c]=d;ans+=w;m++;
    }
    return ans;
}
```

2.3 Shortest Path – Dijkstra with Heap

```
//Using Binary heap
struct edge{int v,w,next;};
void init(int g[],edge e[],int &n,int &m)
{
    cin>>n>>m;
    memset(g,255,sizeof(*g)*n); //-1 indicates end of link
    for(int i=0,j=0;i<m;i++)
    {
        int a,b,c;
        cin>>a>>b>>c; //Vertex numbered in [0,n)
        e[j].v=b;e[j].w=c;e[j].next=g[a];g[a]=j++;
        e[j].v=a;e[j].w=c;e[j].next=g[b];g[b]=j++; //Opposite direction
    }
}
//Return dis[t]. Set t=-1 if all dis[] wanted.
int Dijkstra(int g[],int n,edge e[],int s,int t,int d[])
{
    int h[maxn+1],r[maxn],hn=1,k;
    d[s]=0;h[1]=s;r[s]=1;
    for(int i=0;i<n;i++)
        if(i!=s) {d[i]=INT_MAX;h[++hn]=i;r[i]=hn;}
    while(hn>0&&d[k=BH_Pop(h,r,d,hn)]<INT_MAX)
    {
        if(t==k) return d[t];
        for(int i=g[k];i!=-1;i=e[i].next)
            if(d[e[i].v]>d[k]+e[i].w)
            {
                d[e[i].v]=d[k]+e[i].w;
                BH_Up(h,r,d,e[i].v);
```

```
            }
    }
    return INT_MAX;
}
```

2.4 Shortest Path – Bellman-Ford with Queue

```
//Return false if negative-weight circle found.
//* For Minimum Cost Flow
bool BellmanFord(int g[],edge e[],int n,int s,int dis[])
{
    bool u[maxn]={0};
    int q[maxn],c[maxn]={0},h=0,d=1;
    for(int i=0;i<n;i++) dis[i]=INT_MAX;
    dis[s]=0;u[s]=true;q[0]=s;
    while(h!=d)
    {
        int i=q[h];
        if(++h==n+1) h=0;
        u[i]=false;
        if(c[i]++==n) return false;
        for(int j=g[i],k;j!=-1;j=e[j].next)
            if(dis[k=e[j].v]>dis[i]+e[j].w)
            //* if(e[j].r<j&&dis[k=e[j].v]>dis[i]-e[j].w)
            {
                dis[k]=dis[i]+e[j].w; //* dis[k]=dis[i]-e[j].w;
                if(!u[k]) {u[k]=true;q[d++]=k;if(d==n+1) d=0;}
            }
    }
    return true;
}
```

2.5 Shortest Path – Floyd-Warshall

```
void FloydWarshall(int g[][maxn],int n) //g[a][b]==-1: no edge from a to b
{
    for(int i=0;i<n;i++)
        for(int j=0;j<n;j++)
            for(int k=0;k<n;k++)
                if(g[j][i]!=-1&&g[i][k]!=-1&&(g[j][k]==-1||g[j][k]>g[j][i]+g[i][k]))
                    g[j][k]=g[j][i]+g[i][k];
}
```

2.6 Maximum Flow – Edmonds-Karp in Link

```
//Add a reversal edge with capacity 0 for each edge in a directed graph.
//edge::r=Reversal edge's ID
struct edge {int v,c,f,r,next;};
int EdmondsKarp(int g[],edge e[],int n,int src,int dst)
{
    int d[maxn]={},p[maxn],h[maxn],r[maxn],c[maxn]={n};
    int flow=0,delta=INT_MAX,v=src;
    memcpy(r,g,sizeof(r));
    p[src]=-1;
    while(d[src]<n)
    {
        bool flag=true;
        h[v]=delta;
        for(int i=r[v];i!=-1;i=e[i].next)
            if(e[i].c>e[i].f&&d[v]==d[e[i].v]+1)
            {
                if(delta>e[i].c-e[i].f) delta=e[i].c-e[i].f;
                flag=false;
                r[v]=i;
                p[e[i].v]=e[i].r;
                v=e[i].v;
                break;
            }
        if(flag)
        {
            int t=n-1;
            for(int i=g[v];i!=-1;i=e[i].next)
                if(t>d[e[i].v]&&e[i].f<e[i].c) {t=d[e[i].v];r[v]=i;}
```

```
            if(--c[d[v]]==0) break;
            d[v]=t+1;
            c[d[v]]++;
            if(p[v]!=-1) {v=e[p[v]].v;delta=h[v];}
        }
        else if(v==dst)
        {
            flow+=delta;
            while(p[v]!=-1)
            {
                e[p[v]].f-=delta;
                e[e[p[v]].r].f+=delta;
                v=e[p[v]].v;
            }
            delta=INT_MAX;
        }
    }
    return flow;
}
```

2.7 Maximum Flow – Edmonds-Karp in Matrix

```
int EdmondsKarp(int g[][maxn],int n,int src,int dst,int f[][maxn])
{
    int d[maxn]={},p[maxn],r[maxn]={},c[maxn]={n};
    int flow=0,delta=INT_MAX,h[maxn],v=src;
    memset(f,0,sizeof(*f)*n);
    p[src]=-1;
    while(d[src]<n)
    {
        bool flag=true;;
        h[v]=delta;
        for(int i=r[v];i<n;i++)
            if(g[v][i]>f[v][i]&&d[v]==d[i]+1)
            {
                if(delta>g[v][i]-f[v][i]) delta=g[v][i]-f[v][i];
                flag=false;
                r[v]=i;p[i]=v;v=i;
                break;
            }
        if(flag)
        {
            int t=n-1;
            for(int i=0;i<n;i++)
                if(t>d[i]&&f[v][i]<g[v][i]) {t=d[i];r[v]=i;}
            if(--c[d[v]]==0) break;
            d[v]=t+1;
            c[d[v]]++;
            if(p[v]!=-1) {v=p[v];delta=h[v];}
        }
        else if(v==dst)
        {
            flow+=delta;
            while(p[v]!=-1)
            {
                f[v][p[v]]-=delta;
                f[p[v]][v]+=delta;
                v=p[v];
            }
            delta=INT_MAX;
        }
    }
    return flow;
}
```

2.8 Minimum Cost Flow

```
//Replace each undirected edge with two opposite directed edge.
struct edge {int v,c,w,f,r,next;};
void AddEdge(int g[],edge e[],int &m,int u,int v,int c,int w)
{
    e[m].f=0;e[m].v=v;e[m].c=c;e[m].w=+w;e[m].r=m+1;e[m].next=g[u];g[u]=m++;
    e[m].f=0;e[m].v=u;e[m].c=0;e[m].w=-w;e[m].r=m-1;e[m].next=g[v];g[v]=m++;
```

```
}
//For sparse graph
int MinCost(int g[],edge e[],int n,int src,int dst,int &flow)
{
    static int q[maxn],w[maxn],c[maxn],f[maxn],p[maxn];
    static bool u[maxn];
    int cost=0;
    flow=0;
    while(true)
    {
        int h=0,d=1;
        memset(c,0,sizeof(c));
        memset(u,0,sizeof(u));
        for(int i=0;i<n;i++) w[i]=INT_MAX;
        q[0]=src;
        w[src]=0;
        f[src]=INT_MAX;
        while(h!=d)
        {
            int i=q[h++];
            if(h==n) h=0;
            if(++c[i]==n) {flow=0;return 0;}
            u[i]=false;
            for(int j=g[i];j!=-1;j=e[j].next)
                if(e[j].c>e[j].f&&w[e[j].v]>w[i]+e[j].w)
                {
                    w[e[j].v]=w[i]+e[j].w;
                    f[e[j].v]=min(f[i],e[j].c-e[j].f);
                    p[e[j].v]=e[j].r;
                    if(!u[e[j].v])
                    {
                        q[d++]=e[j].v;
                        u[e[j].v]=true;
                        if(d==n) d=0;
                    }
                }
        }
        if(w[dst]==INT_MAX) return cost;
        flow+=f[dst];
        cost+=f[dst]*w[dst];
        for(int i=dst;i!=src;i=e[p[i]].v)
        {
            e[p[i]].f-=f[dst];
            e[e[p[i]].r].f+=f[dst];
        }
    }
}
//For dense graph, using Bellman-Ford
int MinCost(int g[],edge e[],int n,int src,int dst,int &flow)
{
    static int v[maxn],q[maxn],f[maxn],p[maxn];
    static bool u[maxn];
    int cost=0;
    flow=0;
    if(!BellmanFord(g,e,n,dst,v)||v[src]==INT_MAX) return 0;
    while(true)
    {
        while(true)
        {
            memset(u,0,sizeof(u));
            u[src]=true;
            f[src]=INT_MAX;
            q[0]=src;
            int h=0,d=1;
            while(h<d&&!u[dst])
            {
                int i=q[h++];
                for(int j=g[i];j!=-1;j=e[j].next)
                    if(!u[e[j].v]&&e[j].c>e[j].f)
                    {
                        if(v[e[j].v]+e[j].w==v[i])
                        {
```

```
                            f[e[j].v]=min(f[i],e[j].c-e[j].f);
                            u[e[j].v]=true;
                            p[e[j].v]=e[j].r;
                            q[d++]=e[j].v;
                        }
                    }
            }
            if(!u[dst]) break;
            flow+=f[dst];
            cost+=f[dst]*v[src];
            for(int i=dst;i!=src;i=e[p[i]].v)
            {
                e[p[i]].f-=f[dst];
                e[e[p[i]].r].f+=f[dst];
            }
        }
        int delta=INT_MAX;
        for(int i=0;i<n;i++)
            if(u[i]) for(int j=g[i];j!=-1;j=e[j].next)
                if(e[j].c>e[j].f&&!u[e[j].v]) delta=min(delta,v[e[j].v]+e[j].w-v[i]);
        if(delta==INT_MAX) return cost;
        for(int i=0;i<n;i++) if(u[i]) v[i]+=delta;
    }
}
```

2.9 Maximum Matching on Bipartite Graph – Hungarian

```
bool MaxMatchDFS(bool g[][maxn],int m,int a,int y[],bool u[])
{
    for(int i=0;i<m;i++)
        if(!u[i]&&g[a][i])
        {
            int t=y[i];
            u[i]=true;y[i]=a;
            if(t==-1||MaxMatchDFS(g,m,t,y,u)) return true;
            y[i]=t;
        }
    return false;
}
int MaxMatch(bool g[][maxn],int n,int m,int y[]) //v1[y[i]] matches v2[i]
{
    memset(y,255,sizeof(*y)*m);
    int c=0;
    for(int i=0;i<n;i++)
    {
        bool u[maxn]={0};
        if(MaxMatchDFS(g,m,i,y,u)) c++;
    }
    return c;
}
```

2.10 Minimum Vertex Cover on Bipartite Graph

```
int MinCover(bool g[][maxn],int n,int m,bool a[],bool b[])
{
    int y[maxn],q[maxn],h=0,e=0;
    int c=MaxMatch(g,n,m,y);
    memset(a,0,n*sizeof(*a));
    memset(b,0,m*sizeof(*b));
    for(int i=0;i<m;i++) if(y[i]>=0) a[y[i]]=true;
    for(int i=0;i<n;i++) if(!a[i]) q[e++]=i;
    while(h<e)
    {
        for(int i=0;i<m;i++)
            if(y[i]>=0&&g[q[h]][i])
            {
                b[i]=true;
                if(a[y[i]]) {q[e++]=y[i];a[y[i]]=false;}
            }
        h++;
    }
}
```

2.11 Maximum Cost Perfect Matching on Bipartite Graph – Kuhn-Munkres

```
bool KM_DFS(int g[][maxn],int n,int a,int y[],bool u[],bool v[],int lx[],int ly[])
{
    u[a]=true;
    for(int i=0;i<n;i++)
        if(!v[i]&&g[a][i]==lx[a]+ly[i])
        {
            v[i]=true;
            if(y[i]==-1||KM_DFS(g,n,y[i],y,u,v,lx,ly)) {y[i]=a;return true;}
        }
    return false;
}
//Set g[x][y]=0 if there's no (x,y). n=max(|X|,|Y|).
//Return the sum of maximum weight. y[i] matches i.
int KM(int g[][maxn],int n,int y[])
{
    int lx[maxn],ly[maxn]={},w=0;
    memset(y,255,sizeof(*y)*n);
    for(int i=0;i<n;i++)
    {
        lx[i]=INT_MIN;
        for(int j=0;j<n;j++)
            if(lx[i]<g[i][j]) lx[i]=g[i][j];
    }
    for(int i=0;i<n;i++)
    {
        bool u[maxn]={},v[maxn]={};
        while(!KM_DFS(g,n,i,y,u,v,lx,ly))
        {
            int d=INT_MAX;
            for(int i=0;i<n;i++)
                if(u[i])
                    for(int j=0;j<n;j++)
                        if(!v[j]&&d>lx[i]+ly[j]-g[i][j]) d=lx[i]+ly[j]-g[i][j];
            for(int i=0;i<n;i++)
            {
                if(u[i]) {lx[i]-=d;u[i]=false;}
                if(v[i]) {ly[i]+=d;v[i]=false;}
            }
        }
    }
    for(int i=0;i<n;i++) w+=g[y[i]][i];
    return w;
}
```

2.12 Lowest Common Ancestor – Tarjan off-line

```
struct edge {int v,next;};
//w==u.parent, g and e indicate the tree, q and p indicate the queries, a and c=={}
void Tarjan(int u,int w,int g[],edge e[],int q[],edge p[],int f[],int a[],bool c[])
{
    MakeSet(f,u);
    a[u]=u;
    for(int i=g[u];i!=-1;i=e[i].next)
        if(e[i].v!=w)
        {
            Tarjan(e[i].v,u,g,e,q,p,f,a,c);
            Union(f,e[i].v,u);
            a[Find(f,u)]=u;
        }
    c[u]=true;
    for(int i=q[u];i!=-1;i=p[i].next)
        if(p[i].v!=w&&c[p[i].v])
            cout<<"LCA("<<u<<','<<p[i].v<<")="<<a[Find(f,p[i].v)]<<endl;
}
```

2.13 Strongly Connected Component – DFS

```
struct edge{int v,next;};
void StrongDFS(int a,int g[],edge e[],int u[],int &b)
```

```
{
    u[a]=-1;
    for(int i=g[a];i!=-1;i=e[i].next)
        if(!u[e[i].v]) StrongDFS(e[i].v,g,e,u,b);
    u[a]=++b;
}
void StrongBack(int a,int f[],edge e[],int *r,int b)
{
    r[a]=b;
    for(int i=f[a];i!=-1;i=e[i].next)
        if(r[e[i].v]==-1) StrongBack(e[i].v,f,e,r,b);
}
//g[]=Origin graph, f[]=Reversed g[]
//Return the number of components. r[] stores IDs for each vertex.
int StrongCC(int g[],int f[],edge e[],int n,int r[])
{
    int u[maxn]={},v[maxn],b=0,m=0;
    memset(r,255,sizeof(*r)*n);
    for(int i=0;i<n;i++)
        if(!u[i]) StrongDFS(i,g,e,u,b);
    for(int i=0;i<n;i++) v[u[i]-1]=i;
    for(int i=n-1;i>=0;i--)
        if(r[v[i]]==-1) StrongBack(v[i],f,e,r,m++);
    return m;
}
```

2.14 Cut-vertex – DFS

```
struct edge {int v,next;};
void CV_Dfs(int g[],edge e[],int a,int b,int &c,int u[],int v[],bool r[],int &m)
{
    v[a]=u[a]=++c;
    bool p=false;
    int d=0,j;
    for(int i=g[a];i!=-1;i=e[i].next)
        if((j=e[i].v)!=b)
        {
            if(v[j]) {if(v[a]>u[j]) v[a]=u[j];}
            else
            {
                CV_Dfs(g,e,j,a,c,u,v,r,m);
                d++;
                if(u[a]<=v[j]) p=true;
                if(v[a]>v[j]) v[a]=v[j];
            }
        }
    if(!r[a]&&((b==-1&&d>1)||(b>=0&&p))) {r[a]=true;m++;}
}
//m=number of cut-vertices. r[]=list of cut-vertices.
void CutVertex(int g[],int n,edge e[],bool r[],int &m)
{
    int c=0;
    m=0;
    int u[maxn]={},v[maxn]={};
    memset(r,0,sizeof(*r)*n);
    for(int i=0;i<n;i++)
        if(!u[i]) CV_Dfs(g,e,i,-1,c,u,v,r,m);
}
```

2.15 Cut-edge – DFS

```
struct edge {int v,next;};
struct info {int u,v;};
bool CE_cmp(const info &a,const info &b)
{
    return a.u<b.u||(a.u==b.u&&a.v<b.v);
}
void CE_Dfs(int g[],edge e[],int a,int b,int &c,int u[],int v[],info r[],int &m)
{
    v[a]=u[a]=++c;
    for(int i=g[a],j;i!=-1;i=e[i].next)
        if((j=e[i].v)!=b)
        {
```

```
            if(v[j]) {if(v[a]>u[j]) v[a]=u[j];}
            else
            {
                CE_Dfs(g,e,j,a,c,u,v,r,m);
                if(v[a]>v[j]) v[a]=v[j];
            }
            if(u[a]<v[j]) {r[m].u=a;r[m].v=j;m++;}
        }
}
//m=number of cut-edges. r[]=list of cut-edges.
void CutEdge(int g[],int n,edge e[],info r[],int &m)
{
    int c=0,j=0;
    m=0;
    int u[maxn]={},v[maxn]={};
    for(int i=0;i<n;i++)
        if(!u[i]) CE_Dfs(g,e,i,-1,c,u,v,r,m);
    for(int i=0;i<m;i++)
        if(r[i].u>r[i].v) swap(r[i].u,r[i].v);
    if(m==0) return;
    sort(r,r+m,CE_cmp);
    bool p=true;
    for(int i=1;i<m;i++)
        if(r[i].u!=r[j].u||r[i].v!=r[j].v)
        {
            if(p) j++; else p=true;
            r[j]=r[i];
        }
        else p=false;
    m=j+(p?1:0);
}
```

2.16 Euler Path – DFS

```
//Check the graph by yourself. Set a=source, m=0 for the first call. s[]=Reversed
path.
void EulerPath(bool g[][maxn],int n,int a,int s[],int &m)
{
    for(int i=0;i<n;i++)
        if(g[a][i])
        {
            g[a][i]=false;
            g[i][a]=false; //Remove this line for directed graph.
            EulerPath(g,n,i,s,m);
            s[m++]=i;
        }
}
```

2.17 Topological Sort – DFS

```
struct edge {int v,next;};
bool TopSortDFS(int v,edge e[],int g[],int u[],int &m)
{
    u[v]=-2;
    for(int i=g[v];i!=-1;i=e[i].next)
        if(u[e[i].v]==-2) return false;
        else if(u[e[i].v]==-1&&!TopSortDFS(e[i].v,e,g,u,m)) return false;
    u[v]=--m;
    return true;
}
bool TopSort(edge e[],int g[],int n,int list[])
{
    static int u[maxn];
    int m=n;
    memset(u,255,sizeof(u));
    for(int i=0;i<n;i++)
        if(u[i]==-1&&!TopSortDFS(i,e,g,u,m)) return false;
    for(int i=0;i<n;i++) list[u[i]]=i;
    return true;
}
```

2.18 Tree Isomorphism

```
struct edge {int v,next;};
struct label {int n,v,p,r,h;short l[maxl];};
bool TreeIsm_cmp(const label &a,const label &b)
{
    if(a.n<b.n) return true;
    if(a.n>b.n) return false;
    return memcmp(a.l,b.l,a.n*sizeof(*a.l))<0;
}
//Two trees are isomorphous only when list[link[r1]].l==list[link[r2]].l
void TreeIsm(int g[],edge e[],int root,label list[],int link[])
{
    int h=0,n=1;
    memset(link,255,sizeof(*link)*maxn);
    memset(list,0,sizeof(*list)*maxn);
    list[0].v=root;
    link[root]=0;
    while(h<n)
    {
        for(int i=g[list[h].v];i!=-1;i=e[i].next)
            if(link[e[i].v]==-1)
            {
                list[n].v=e[i].v;
                list[n].p=list[h].v;
                list[n].h=list[h].h+1;
                link[e[i].v]=n;
                n++;
            }
        h++;
    }
    for(int i=n-1,j=n-1;i>0;i=j)
    {
        while(j>=0&&list[j].h==list[i].h) j--;
        sort(list+j+1,list+i+1,TreeIsm_cmp);
        for(int k=j+1,r=-1;k<=i;k++)
        {
            int p=link[list[k].p];
            if(k==j+1||TreeIsm_cmp(list[k-1],list[k])) r++;
            if(list[p].n<=r) list[p].n=r+1;
            list[p].l[r]++;
            list[k].r=r;
            link[list[k].v]=k;
        }
    }
}
```

2.19 Minimum-cut – Stoer-Wagner in Matrix

```
int StoerWagner(int g[][maxn],int n,int &src,int &dst)
{
    int v[maxn],cut=INT_MAX;
    for(int i=0;i<n;i++) v[i]=i;
    for(int m=n;m>1;m--)
    {
        int s=-1,t=0,w[maxn]={},z=INT_MIN;
        bool u[maxn]={};
        for(int i=1;i<m;i++)
        {
            z=INT_MIN;
            u[v[s=t]]=true;
            for(int j=0;j<m;j++)
                if(!u[v[j]])
                {
                    w[v[j]]+=g[v[s]][v[j]];
                    if(z<w[v[j]]) z=w[v[t=j]];
                }
        }
        if(cut>z) {cut=z;src=v[s];dst=v[t];}
        for(int i=0;i<m;i++) g[v[t]][v[i]]+=g[v[s]][v[i]];
        for(int i=0;i<m;i++) g[v[i]][v[t]]=g[v[t]][v[i]];
        v[s]=v[m-1];
```

```
    }
    return cut;
}
```

2.20 Tree Center – DFS

```
struct edge {int v,w,next;};
void Depth(int g[],edge e[],int u,int v,int d[],int k[])
{
    d[v]=0;
    for(int i=g[v];i!=-1;i=e[i].next)
    {
        if(e[i].v==u) continue;
        Depth(g,e,v,e[i].v,d,k);
        if(d[v]<d[e[i].v]+e[i].w)
        {
            d[v]=d[e[i].v]+e[i].w;
            k[v]=e[i].v;
        }
    }
}
void Distance(int g[],edge e[],int u,int v,int d[],int k[],int l)
{
    if(d[v]<l) d[v]=l;
    for(int i=g[v];i!=-1;i=e[i].next)
    {
        if(e[i].v==u||e[i].v==k[v]) continue;
        if(l<d[e[i].v]+e[i].w) l=d[e[i].v]+e[i].w;
    }
    for(int i=g[v];i!=-1;i=e[i].next)
    {
        if(e[i].v==u) continue;
        Distance(g,e,v,e[i].v,d,k,(e[i].v==k[v]?l:d[v])+e[i].w);
    }
}
void TreeCenter(int g[],int n,edge e[],int &r,int &c1,int &c2)
{
    static int d[maxn],k[maxn];
    Depth(g,e,-1,0,d,k);
    Distance(g,e,-1,0,d,k,0);
    r=INT_MAX;
    for(int i=0;i<n;i++)
        if(r==d[i]) c2=i;
        else if(r>d[i]) r=d[c2=c1=i];
}
```

2.21 Minimum Vertex Cover

```
bool IsCover(bool g[][maxn],int n,bool cover[],int degree[],int m)
{
    int v[2]={-1},match=0;
    bool u[maxn];
    memcpy(u,cover,sizeof(u));
    for(int i=0;i<n;i++)
    {
        if(cover[i]) continue;
        for(int j=0;j<i;j++)
        {
            if(cover[j]||!g[i][j]) continue;
            if(!u[i]&&!u[j])
            {
                u[i]=u[j]=true;
                match++;
            }
            if(v[0]==-1||degree[v[0]]+degree[v[1]]<degree[i]+degree[j])
            {
                v[0]=i;
                v[1]=j;
            }
        }
    }
    if(v[0]==-1) return true;
    if(m<match) return false;
```

```
    for(int k=0;k<2;k++)
    {
        for(int i=0;i<n;i++) if(g[v[k]][i]) degree[i]--;
        cover[v[k]]=true;
        if(IsCover(g,n,cover,degree,m-1)) return true;
        cover[v[k]]=false;
        for(int i=0;i<n;i++) if(g[v[k]][i]) degree[i]++;
    }
    return false;
}
int MinCover(bool g[][maxn],int n,bool cover[])
{
    int degree[maxn]={};
    for(int i=0;i<n;i++)
        for(int j=0;j<n;j++)
            if(i==j) g[i][j]=false;
            else if(g[i][j]) degree[i]++;
    int upper=0;
    int d[maxn];
    memcpy(d,degree,sizeof(d));
    memset(cover,0,sizeof(*cover)*n);
    for(int i=1;i<n;i++)
    {
        int k=-1;
        for(int j=0;j<n;j++)
            if(!cover[j]&&(k==-1||d[k]<d[j])) k=j;
        if(k==-1) break;
        upper++;
        for(int j=0;j<n;j++)
            if(!cover[j]&&g[k][j]) d[j]--;
    }
    for(int i=upper-1;i>=0;i--)
    {
        bool c[maxn]={};
        memcpy(d,degree,sizeof(d));
        if(!IsCover(g,n,c,d,i)) return i+1;
        memcpy(cover,c,sizeof(c));
    }
    return 0;
}
```

2.22 Minimum Dominating Vertex Set

```
bool IsDominate(bool g[][maxn],int n,bool dom[],int cover[],int degree[],int m)
{
    int target=-1,uncover[maxn]={},tmp=0;
    for(int i=0;i<n;i++)
    {
        if(cover[i]) continue;
        uncover[i]++;
        tmp++;
        for(int j=0;j<n;j++)
            if(g[i][j]) uncover[j]++;
        if(target==-1||degree[target]>degree[i]) target=i;
    }
    if(target==-1) return true;
    if(m==0) return false;
    sort(uncover,uncover+n);
    for(int i=1;i<=m;i++) tmp-=uncover[n-i];
    if(tmp>0) return false;
    for(int i=0;i<n;i++)
    {
        if(!(degree[target]==0&&target==i)&&(!g[target][i]||dom[i])) continue;
        for(int j=0;j<n;j++)
            if(g[i][j]) {cover[j]++;degree[j]--;}
        cover[i]++;
        dom[i]=true;
        if(IsDominate(g,n,dom,cover,degree,m-1)) return true;
        dom[i]=false;
        cover[i]--;
        for(int j=0;j<n;j++)
            if(g[i][j]) {cover[j]--;degree[j]++;}
```

```
    }
    return false;
}
int MinDominate(bool g[][maxn],int n,bool dom[])
{
    int degree[maxn]={};
    memset(dom,0,sizeof(*dom)*n);
    for(int i=0;i<n;i++)
        for(int j=0;j<n;j++)
            if(i==j) g[i][j]=false;
            else if(g[i][j]) degree[i]++;
    for(int i=0;i<n;i++)
    {
        int cover[maxn]={};
        if(IsDominate(g,n,dom,cover,degree,i)) return i;
    }
    memset(dom,true,sizeof(*dom)*n);
    return n;
}
```

3 Number Theory

3.1 Extended Euclidean

```
//Solve ax+by=(a,b)
int gcd(int a,int b,int &x,int &y)
{
    if(b==0) {x=1;y=0;return a;}
    int r=gcd(b,a%b,y,x);
    y-=a/b*x;
    return r;
}
```

3.2 Chinese Remainder Theorem

```
//Solve x=ai(mod mi), for any i and j, (mi,mj)==1.
//Return x0 in [0,M). All solutions are x=x0+tM.
int Chinese(int a[],int m[],int n)
{
    int s=1,t,ans=0,p,q;
    for(int i=0;i<n;i++) s*=m[i];
    for(int i=0;i<n;i++)
    {
        t=s/m[i];
        gcd(t,m[i],p,q);
        ans=(ans+t*p*a[i])%s;
    }
    if(ans<0) ans+=s;
    return ans;
}
```

3.3 Linear congruences

```
//Solve x=ai(mod mi), for any i and j, (mi,mj)|ai-aj.
//Return x0 in [0,[m1..mn]). All solutions are x=x0+t[m1..mn].
int LinearCon(int a[],int m[],int n)
{
    int u=a[0],v=m[0],p,q,r,t;
    for(int i=1;i<n;i++)
    {
        r=gcd(v,m[i],p,q);
        t=v;
        v=v/r*m[i];
        u=((a[i]-u)/r*p*t+u)%v;
    }
    if(u<0) u+=v;
    return u;
}
```

3.4 Prime generator

```
void PrimeGen(int p[],int &n)
```

```
{
    static bool q[maxp+1]={0};
    for(int i=2;i*i<=maxp;i++)
        if(!q[i]) for(int j=i*2;j<=maxp;j+=i) q[j]=true;
    n=0;
    for(int i=2;i<=maxp;i++) if(!q[i]) p[n++]=i;
}
```

3.5 Euler Totient function

```
int phi(int a,int p[])
{
    int b=a;
    for(int i=0;p[i]*p[i]<=a;i++)
        if(a%p[i]==0)
        {
            b=b/p[i]*(p[i]-1);
            do a/=p[i]; while(a%p[i]==0);
        }
    if(a>1) b=b/a*(a-1);
    return b;
}
```

3.6 Farey Sequence generator

```
//|Fn|=1+sum(phi(m),m=1..n)
int FareySeq(int n,int s[][2])
{
    int m=1,a=0,b=1,c=1,d=n,e,f;
    s[0][0]=0;
    s[0][1]=1;
    while(c<n)
    {
        int k=(n+b)/d;
        e=k*c-a;f=k*d-b;
        a=c;b=d;c=e;d=f;
        s[m][0]=a;s[m][1]=b;
        m++;
    }
    return m;
}
```

3.7 Exponentiation by squaring

```
typedef unsigned long long uint64_t;
uint64_t mul_mod(uint64_t a,uint64_t b,uint64_t m)
{
    uint64_t ah=a>>32,al=a&0xffffffffull,bh=b>>32,bl=b&0xffffffffull;
    uint64_t rh=ah*bh,rl=al*bl,x=ah*bl,y=al*bh;
    rh+=(x>>32)+(y>>32);
    x<<=32;y<<=32;
    rl+=x;
    if(rl<x) rh++;
    rl+=y;
    if(rl<y) rh++;
    if(rh>=m) rh%=m;
    for(int i=0;i<64;i++)
    {
        rh<<=1;
        if(rl&(1ull<<63)) rh|=1;
        rl<<=1;
        if(rh>=m) rh-=m;
    }
    return rh;
}
uint64_t pow_mod(uint64_t a,uint64_t n,uint64_t m)
{
    uint64_t p=1;
    for(uint64_t i=1;i<=n;i<<=1)
    {
        if(i&n) p=mul_mod(p,a,m);
        a=mul_mod(a,a,m);
    }
```

```
    return p;
}
```

3.8 Primality Test – Miller-Rabin

```
bool MillerRabin(uint64_t n,int tries) //Return true if prime
{
    if(n==1) return false;
    if(n==2||n==3) return true;
    if(!(n&1)) return false;
    uint64_t d=n-1;
    int s=0;
    while(!(d&1)) {d>>=1;s++;}
    while(tries--)
    {
        uint64_t x=pow_mod(rand()%(n-3)+2,d,n),y;
        for(int j=0;j<s;j++)
        {
            y=mul_mod(x,x,n);
            if(y==1&&x!=1&&x!=n-1) return false;
            x=y;
        }
        if(x!=1) return false;
    }
    return true;
}
```

3.9 Integer Factorization – Pollard's Rho

```
uint64_t gcd(uint64_t a,uint64_t b)
{
    while(b)
    {
        uint64_t t=a%b;
        a=b;
        b=t;
    }
    return a;
}
uint64_t PollardRho(uint64_t n) //n shouldn't be prime
{
    if(!(n&1)) return 2;
    while(true)
    {
        uint64_t x=(uint64_t)rand()%n,y=x;
        uint64_t c=rand()%n;
        if(c==0||c==2) c=1;
        for(int i=1,k=2;;i++)
        {
            x=mul_mod(x,x,n);
            if(x>=c) x-=c; else x+=n-c;
            if(x==n) x=0;
            if(x==0) x=n-1; else x--;
            uint64_t d=gcd(x>y?x-y:y-x,n);
            if(d==n) break;
            if(d!=1) return d;
            if(i==k) {y=x;k<<=1;}
        }
    }
}
```

3.10 Primitive Root

```
int primitive_root(int m,int p[])
{
    //only 2, 4, p^n, 2p^n have primitive root
    if(m==1) return 0;
    if(m==2) return 1;
    if(m==4) return 3;
    int t=m;
    if((t&1)==0) t>>=1;
    for(int i=0;p[i]*p[i]<=t;i++)
    {
```

```
        if(t%p[i]) continue;
        do t/=p[i]; while(t%p[i]==0);
        if(t>1||p[i]==2) return 0;
    }
    int x=phi(m,p),y=x,f[32],n=0;
    for(int i=0;p[i]*p[i]<=y;i++)
    {
        if(y%p[i]) continue;
        do y/=p[i]; while(y%p[i]==0);
        f[n++]=p[i];
    }
    if(y>1) f[n++]=y;
    for(int i=1;i<m;i++)
    {
        if(gcd(i,m)>1) continue;
        bool flag=true;
        for(int j=0;j<n;j++)
            if(pow_mod(i,x/f[j],m)==1)
            {
                flag=false;
                break;
            }
        if(flag) return i;
    }
    return 0;
}
```

3.11 Descrete Logarithm – Baby-step Giant-step

```
int dlog(int a,int b,int m,int p[]) //Solve a^x=b(mod m)
{
    hash_map<int,int> hash;
    int n=phi(m,p),k=sqrt(n);
    for(int i=0,t=1;i<k;i++)
    {
        hash[t]=i;
        t=(long long)t*(long long)a%m;
    }
    int c=pow_mod(a,n-k,m);
    for(int i=0;i*k<n;i++)
    {
        if(hash.find(b)!=hash.end()) return i*k+hash[b];
        b=(long long)b*(long long)c%m;
    }
    return -1;
}
```

4 Computational Geometry

4.1 Basic functions

```
const double PI=3.141592653589793,EPS=1e-9,INF=1e100;
struct vect
{
    double x,y;
    vect(double a=0.0,double b=0.0):x(a),y(b) {}
    vect operator+(vect a)
    {
        return vect(x+a.x,y+a.y);
    }
    vect operator-(vect a)
    {
        return vect(x-a.x,y-a.y);
    }
    double operator*(vect a) //Dot Product
    {
        return x*a.x+y*a.y;
    }
    double operator/(vect a) //Cross Product
    {
        return x*a.y-y*a.x;
```

```
    }
    vect operator*(double a)
    {
        return vect(x*a,y*a);
    }
    vect operator/(double a)
    {
        return vect(x/a,y/a);
    }
    vect operator-()
    {
        return vect(-x,-y);
    }
    bool operator==(vect a)
    {
        return fabs(x-a.x)<EPS&&fabs(y-a.y)<EPS;
    }
    bool operator!=(vect a)
    {
        return fabs(x-a.x)>EPS||fabs(y-a.y)>EPS;
    }
    double length()
    {
        return sqrt(x*x+y*y);
    }
    bool iszero()
    {
        return fabs(x)<EPS&&fabs(y)<EPS;
    }
    vect rotate(double a)
    {
        double c=cos(a),s=sin(a);
        return vect(x*c-y*s,x*s+y*c);
    }
};
double radian(vect a,vect b) //[0,PI*2)
{
    double r=atan2(a/b,a*b);
    return r<-EPS?r+PI*2:r;
}
int dbcmp(double a,double b)
{
    return b-a>EPS?-1:a-b>EPS;
}
bool dbzero(double a)
{
    return fabs(a)<EPS;
}
double angle(double a,double b)
{
    double c=b-a;
    while(c<0.0) c+=PI*2.0;
    return c;
}
//Return distance between p and (a,b), from p to a+(b-a)*s
double point2segment(vect &p,vect &a,vect &b,double &s)
{
    if((p-a)*(b-a)<EPS)
    {
        s=0.0;
        return (p-a).length();
    }
    if((p-b)*(a-b)<EPS)
    {
        s=1.0;
        return (p-b).length();
    }
    s=((p-a)*(b-a))/((a-b)*(a-b));
    return fabs((p-a)/(b-a))/(a-b).length();
}
```

4.2 Convex Hull – Graham Scan

```
vect *G;
bool Graham_cmp(vect *a,vect *b)
{
    double r=(*a-*G)/(*b-*G);
    if(dbzero(r)) return a->y>b->y+EPS||(dbzero(a->y-b->y)&&a->x>b->x);
    else return r>0.0;
}
//s[] and m indicate the list of points of the Convex Hull, counterclockwise
void Graham(vect p[],int n,vect *s[],int &m)
{
    static vect *w[maxn];
    G=p;
    for(int i=1;i<n;i++)
        if(G->y>p[i].y||(dbzero(G->y-p[i].y)&&G->x>p[i].x)) {w[i-1]=G;G=p+i;}
        else w[i-1]=p+i;
    sort(w,w+n-1,Graham_cmp);
    for(int i=0;i<n-2;i++)
        if(!dbzero((*w[i]-*G)/(*w[i+1]-*G))) {reverse(w,w+i+1);break;}
    w[n-1]=G;s[0]=G;s[1]=w[0];m=1;
    for(int i=1;i<n;i++)
    {
        while(m>0&&((*s[m-1]-*s[m])/(*w[i]-*s[m])>EPS||*w[i]==*s[m])) m--;
        s[++m]=w[i];
    }
    //Remove below lines to allow more than two points on the same edge
    int z=1;
    for(int i=1;i<m;i++)
        if((*s[i+1]-*s[i])/(*s[z-1]-*s[i])>EPS||(*s[i+1]-*s[i])*(*s[z-1]-*s[i])>EPS)
            s[z++]=s[i];
    s[m=z]=s[0];
}
```

4.3 Convex Hull – Rotating Calipers

```
//All points are ordered conterclockwise. No three points lie on the same edge.
#define next(_i,_n) ((_i)+1<_n?(_i)+1:0)
#define prev(_i,_n) ((_i)-1>=0?(_i)-1:_n-1)
```

4.3.1 Minimum Distance

```
#define CheckMinDist(_i,_j,_k) \
{ \
    double s,tmp=_k?point2segment(c1[_i],c2[_j],c2[next(_j,n2)],s) \
                   :point2segment(c2[_j],c1[_i],c1[next(_i,n1)],s); \
    if(dist>tmp) {dist=tmp;p1=_i;p2=_j;e=_k;} \
}
//Minimum distance is indicated by e?(p1 and (p2,next(p2))):(p2 and (p1,next(p1)))
double Convex_MinDist(vect c1[],int n1,vect c2[],int n2,int &p1,int &p2,bool &e)
{
    double dist=INF;
    if(n1==1)
    {
        for(int i=0;i<n2;i++) CheckMinDist(0,i,true);
        return dist;
    }
    if(n2==1)
    {
        for(int i=0;i<n1;i++) CheckMinDist(i,0,false);
        return dist;
    }
    int q1=-1,q2=-1;
    vect v1(INF,INF),v2(-INF,-INF);
    for(int i=0;i<n1;i++)
        if(v1.y>c1[i].y+EPS||v1.y>c1[i].y-EPS&&v1.x>c1[i].x) v1=c1[q1=i];
    for(int i=0;i<n2;i++)
        if(v2.y<c2[i].y-EPS||v2.y<c2[i].y+EPS&&v2.x<c2[i].x) v2=c2[q2=i];
    bool where=radian(vect(1.0,0.0),c1[next(q1,n1)]-v1)
              <radian(vect(-1.0,0.0),c2[next(q2,n2)]-v2);
    for(int i=0;i<(n1+n2)*2;i++)
    {
        int r1=next(q1,n1),r2=next(q2,n2);
```

```
        if(where)
        {
            vect v=c1[r1]-c1[q1];
            double a1=radian(v,c1[next(r1,n1)]-c1[r1]),a2=radian(-v,c2[r2]-c2[q2]);
            CheckMinDist(q1,q2,false);
            if(a2<EPS)
            {
                CheckMinDist(q1,r2,false);
                CheckMinDist(q1,q2,true);
                CheckMinDist(r1,q2,true);
                q2=r2;
                r2=next(q2,n2);
                a2=radian(-v,c2[r2]-c2[q2].x);
            }
            q1=r1;
            where=a1<a2;
        }
        else
        {
            vect v=c2[r2]-c2[q2];
            double a1=radian(-v,c1[r1]-c1[q1]),a2=radian(v,c2[next(r2,n2)]-c2[r2]);
            CheckMinDist(q1,q2,true);
            if(a1<EPS)
            {
                CheckMinDist(r1,q2,true);
                CheckMinDist(q1,q2,false);
                CheckMinDist(q1,r2,false);
                q1=r1;
                r1=next(q1,n1);
                a1=radian(-v,c1[r1]-c1[q1]);
            }
            q2=r2;
            where=a1<a2;
        }
    }
    return dist;
}
```

4.3.2 Maximum Distance

```
#define CheckMaxDist(_i,_j) \
{ \
    double tmp=(c1[_i]-c2[_j])*(c1[_i]-c2[_j]); \
    if(dist<tmp) {dist=tmp;p1=_i;p2=_j;} \
}
double Convex_MaxDist(vect c1[],int n1,vect c2[],int n2,int &p1,int &p2)
{
    double dist=0.0;
    if(n1==1) return Convex_MaxDist(c2,n2,c1,n1,p2,p1);
    if(n2==1)
    {
        for(int i=0;i<n1;i++) CheckMaxDist(i,0);
        return sqrt(dist);
    }
    int q1=-1,q2=-1;
    vect v1(INF,INF),v2(-INF,-INF);
    for(int i=0;i<n1;i++)
        if(v1.y>c1[i].y+EPS||v1.y>c1[i].y-EPS&&v1.x>c1[i].x) v1=c1[q1=i];
    for(int i=0;i<n2;i++)
        if(v2.y<c2[i].y-EPS||v2.y<c2[i].y+EPS&&v2.x<c2[i].x) v2=c2[q2=i];
    bool where=radian(vect(1.0,0.0),c1[next(q1,n1)]-v1)
              <radian(vect(-1.0,0.0),c2[next(q2,n2)]-v2);
    for(int i=0;i<(n1+n2)*2;i++)
    {
        int r1=next(q1,n1),r2=next(q2,n2);
        if(where)
        {
            vect v=c1[r1]-c1[q1];
            double a1=radian(v,c1[next(r1,n1)]-c1[r1]),a2=radian(-v,c2[r2]-c2[q2]);
            CheckMaxDist(q1,q2);
            CheckMaxDist(r1,q2);
            if(a2<EPS)
            {
```

```
                CheckMaxDist(q1,r2);
                CheckMaxDist(r1,r2);
                q2=r2;
                r2=next(q2,n2);
                a2=radian(-v,c2[r2]-c2[q2]);
            }
            q1=r1;
            where=a1<a2;
        }
        else
        {
            vect v=c2[r2]-c2[q2];
            double a1=radian(-v,c1[r1]-c1[q1]),a2=radian(v,c2[next(r2,n2)]-c2[r2]);
            CheckMaxDist(q1,q2);
            CheckMaxDist(q1,r2);
            if(a1<EPS)
            {
                CheckMaxDist(r1,q2);
                CheckMaxDist(r1,r2);
                q1=r1;
                r1=next(q1,n1);
                a1=radian(-v,c1[r1]-c1[q1]);
            }
            q2=r2;
            where=a1<a2;
        }
    }
    return sqrt(dist);
}
```

4.3.3 Diameter

```
#define CheckDiameter(_i,_j) \
{ \
    double tmp=(c[_i]-c[_j])*(c[_i]-c[_j]); \
    if(dist<tmp) {dist=tmp;p=_i;q=_j;} \
}
double Convex_Diameter(vect c[],int n,int &p,int &q)
{
    int s=-1,t=-1;
    vect sp(INF,INF),tp(-INF,-INF);
    for(int i=0;i<n;i++)
    {
        if(sp.y>c[i].y+EPS||sp.y>c[i].y-EPS&&sp.x>c[i].x) sp=c[s=i];
        if(tp.y<c[i].y-EPS||tp.y<c[i].y+EPS&&tp.x<c[i].x) tp=c[t=i];
    }
    if(radian(vect(1.0,0.0),c[next(s,n)]-sp)>radian(vect(-1.0,0.0),c[next(t,n)]-tp))
        swap(s,t);
    double dist=0.0;
    for(int i=0;i<n*2;i++)
    {
        int j=next(s,n),k=next(t,n);
        double sa=radian(c[j]-c[s],c[next(j,n)]-c[j]),ta=radian(c[s]-c[j],c[k]-c[t]);
        CheckDiameter(t,s);
        CheckDiameter(t,j);
        if(ta<EPS)
        {
            CheckDiameter(k,s);
            CheckDiameter(k,j);
            t=k;
            k=next(t,n);
            ta=radian(c[s]-c[j],c[k]-c[t]);
        }
        if(sa<ta) s=j; else {s=t;t=j;}
    }
    return sqrt(dist);
}
```

4.3.4 Width

```
//Edge (p,next(p)) and point q indicate the width
double Convex_Width(vect c[],int n,int &p,int &q)
{
    int s=-1,t=-1;
```

```
    vect sp(INF,INF),tp(-INF,-INF);
    for(int i=0;i<n;i++)
    {
        if(sp.y>c[i].y+EPS||sp.y>c[i].y-EPS&&sp.x>c[i].x) sp=c[s=i];
        if(tp.y<c[i].y-EPS||tp.y<c[i].y+EPS&&tp.x<c[i].x) tp=c[t=i];
    }
    if(radian(vect(1.0,0.0),c[next(s,n)]-sp)>radian(vect(-1.0,0.0),c[next(t,n)]-tp))
        swap(s,t);
    double dist=INF;
    for(int i=0;i<n*2;i++)
    {
        int j=next(s,n),k=next(t,n);
        double sa=radian(c[j]-c[s],c[next(j,n)]-c[j]),ta=radian(c[s]-c[j],c[k]-c[t]);
        double tmp=fabs((c[j]-c[s])/(c[t]-c[s]))/(c[j]-c[s]).length();
        if(dist>tmp) {dist=tmp;p=s;q=t;}
        if(ta<EPS)
        {
            t=k;
            k=next(t,n);
            ta=radian(c[s]-c[j],c[k]-c[t]);
        }
        if(sa<ta) s=j; else {s=t;t=j;}
    }
    return dist;
}
```

4.3.5 Common Tangent

```
#define CheckTangent(_i,_j) \
{ \
    vect v=c1[_i]-c2[_j]; \
    double a=v/(c1[prev(_i,n1)]-c1[_i]),b=v/(c1[next(_i,n1)]-c1[_i]); \
    if(a*b>-EPS) \
    { \
        double c=v/(c2[prev(_j,n2)]-c2[_j]),d=v/(c2[next(_j,n2)]-c2[_j]); \
        if(c*d>-EPS&&a*c>-EPS&&a*d>-EPS&&b*c>-EPS&&b*d>-EPS) \
            p.insert(make_pair(_i,_j)); \
    } \
}
void Convex_Tangent(vect c1[],int n1,vect c2[],int n2,set< pair<int,int> > &p)
{
    p.clear();
    if(n1==1)
    {
        for(int i=0;i<n2;i++) CheckTangent(0,i);
        return;
    }
    if(n2==1)
    {
        for(int i=0;i<n1;i++) CheckTangent(i,0);
        return;
    }
    int q1=-1,q2=-1;
    vect v1(INF,INF),v2(INF,INF);
    for(int i=0;i<n1;i++)
        if(v1.y>c1[i].y+EPS||v1.y>c1[i].y-EPS&&v1.x>c1[i].x) v1=c1[q1=i];
    for(int i=0;i<n2;i++)
        if(v2.y>c2[i].y+EPS||v2.y>c2[i].y-EPS&&v2.x>c2[i].x) v2=c2[q2=i];
    bool where=radian(vect(1.0,0.0),c1[next(q1,n1)]-v1)
              <radian(vect(1.0,0.0),c2[next(q2,n2)]-v2);
    for(int i=0;i<(n1+n2)*2;i++)
    {
        int r1=next(q1,n1),r2=next(q2,n2);
        if(where)
        {
            vect v=c1[r1]-c1[q1];
            double a1=radian(v,c1[next(r1,n1)]-c1[r1]),a2=radian(v,c2[r2]-c2[q2]);
            CheckTangent(q1,q2);
            CheckTangent(r1,q2);
            if(a2<EPS)
            {
                CheckTangent(q1,r2);
                CheckTangent(r1,r2);
```

```
                q2=r2;
                r2=next(q2,n2);
                a2=radian(v,c2[r2]-c2[q2]);
            }
            q1=r1;
            where=a1<a2;
        }
        else
        {
            vect v=c2[r2]-c2[q2];
            double a1=radian(v,c1[r1]-c1[q1]),a2=radian(v,c2[next(r2,n2)]-c2[r2]);
            CheckTangent(q1,q2);
            CheckTangent(q1,r2);
            if(a1<EPS)
            {
                CheckTangent(r1,q2);
                CheckTangent(r1,r2);
                q1=r1;
                r1=next(q1,n1);
                a1=radian(v,c1[r1]-c1[q1]);
            }
            q2=r2;
            where=a1<a2;
        }
    }
}
```

4.4 Closest Pair

```
double CP_Search(vect *s,vect *x[],vect *y[],int n,int &a,int &b,double d)
{
    if(n==1) return d;
    vect *z[maxn];
    int m=n>>1,p=0;
    static bool u[maxn];
    for(int i=0;i<m;i++) u[x[i]-s]=true;
    for(int i=0,j=0,k=m;i<n;i++)
        if(u[y[i]-s]) z[j++]=y[i]; else z[k++]=y[i];
    for(int i=0;i<m;i++) u[x[i]-s]=false;
    d=CP_Search(s,x,z,m,a,b,d);
    d=CP_Search(s,x+m,z+m,n-m,a,b,d);
    for(int i=0;i<n;i++)
        if(fabs(y[i]->x-x[m]->x)<d) z[p++]=y[i];
    for(int i=0;i<p;i++)
        for(int j=i+1;j<p&&z[j]->y-z[i]->y<d;j++)
        {
            double tmp=(*z[j]-*z[i]).length();
            if(d>tmp) {d=tmp;a=z[j]-s;b=z[i]-s;}
        }
    return d;
}
bool CP_cmpx(const vect *a,const vect *b)
{
    return a->x<b->x;
}
bool CP_cmpy(const vect *a,const vect *b)
{
    return a->y<b->y;
}
double ClosestPair(vect s[],int n,int &a,int &b)
{
    vect *x[maxn],*y[maxn];
    for(int i=0;i<n;i++) x[i]=y[i]=s+i;
    sort(x,x+n,CP_cmpx);
    sort(y,y+n,CP_cmpy);
    return CP_Search(s,x,y,n,a,b,INF);
}
```

4.5 Intersections

4.5.1 Line-line Intersection

```
//Solve p+u*s=q+v*t
```

```
//Return value: -1=same line,0=no intersection,1=ok
int ll_intersect(vect &p,vect &q,vect &u,vect &v,double &s,double &t)
{
    double d=v.x*u.y-u.x*v.y;
    double d1=v.x*(q.y-p.y)-v.y*(q.x-p.x),d2=u.x*(q.y-p.y)-u.y*(q.x-p.x);
    if(fabs(d)<EPS)
    {
        if(fabs(d1)<EPS)
        {
            s=t=0.0;
            if(fabs(u.x)>EPS) s=(q.x-p.x)/u.x;
            else if(fabs(u.y)>EPS) s=(q.y-p.y)/u.y;
            else if(fabs(v.x)>EPS) t=(p.x-q.x)/v.x;
            else if(fabs(v.y)>EPS) t=(p.y-q.y)/v.y;
            else return 0;
            return fabs(u.x)<EPS&&fabs(u.y)<EPS||fabs(v.x)<EPS&&fabs(v.y)<EPS?1:-1;
        }
        return 0;
    }
    s=d1/d;
    t=d2/d;
    return 1;
}
```

4.5.2 Circle-circle Intersection

```
//Return number of intersections
int cc_intersect(vect p1,double r1,vect p2,double r2,vect &i1,vect &i2)
{
    double dq=(p1-p2)*(p1-p2),rq=r1*r1-r2*r2;
    vect c=(p1+p2)*0.5+(p2-p1)*rq*0.5/dq;
    double dt=2.0*dq*(r1*r1+r2*r2)-dq*dq-rq*rq;
    if(dt<-EPS) return 0;
    if(dt<EPS)
    {
        i1=i2=c;
        return 1;
    }
    dt=sqrt(dt)*0.5/dq;
    i1=vect(c.x+(p2.y-p1.y)*dt,c.y-(p2.x-p1.x)*dt);
    i2=vect(c.x-(p2.y-p1.y)*dt,c.y+(p2.x-p1.x)*dt);
    return 2;
}
```

4.5.3 Circle-line Intersection

```
//Return number intersections. Intersections are q+us and q+ut.
int cl_intersect(vect p,double r,vect q,vect u,double &s,double &t)
{
    double uq=u*u;
    double d=uq*r*r-(u.x*(q.y-p.y)-u.y*(q.x-p.x))*(u.x*(q.y-p.y)-u.y*(q.x-p.x));
    if(d<-EPS) return 0;
    s=t=-u*(q-p)/uq;
    if(d<EPS) return 1;
    d=sqrt(d)/uq;
    s-=d;
    t+=d;
    return 2;
}
```

4.6 Half-plane Intersection

```
struct plane
{
    double angle,dist; //Distance from (0,0) to the line in direction of angle
};
bool intersect_cmp(const plane *a,const plane *b)
{
    return a->angle<b->angle||(fabs(a->angle-b->angle)<EPS&&a->dist>b->dist);
}
double dist(plane &a,plane &b,double angle)
{
    if(fabs(fabs(a.angle-b.angle)-PI)<EPS) return INF;
    return (a.dist*sin(b.angle-angle)-b.dist*sin(a.angle-angle))/sin(b.angle-a.angle);
```

```
}
int intersect(plane p[],int n,int r[])
{
    static plane *s[maxn];
    for(int i=0;i<4;i++)
    {
        p[n+i].angle=PI*i/2;
        p[n+i].dist=-INF;
    }
    n+=4;
    for(int i=0;i<n;i++) s[i]=p+i;
    sort(s,s+n,intersect_cmp);
    int m=1;
    for(int i=1;i<n;i++)
        if(s[i]->angle-s[m-1]->angle>EPS) s[m++]=s[i];
    for(int i=0,j=0;;i++)
    {
        while(j<n&&s[j]->angle-s[i]->angle<PI-EPS) j++;
        if(j>=n) break;
        if(s[j]->angle-s[i]->angle<PI+EPS&&s[j]->dist+s[i]->dist>-EPS) return 0;
    }
    r[0]=s[0]-p;
    if(m==1) return 1;
    r[1]=s[1]-p;
    if(m==2) return 2;
    int h=0,e=2;
    for(int i=2;i<m;i++)
    {
        while(e-h>1&&dist(p[r[e-2]],p[r[e-1]],s[i]->angle)<s[i]->dist+EPS) e--;
        while(e-h>1&&dist(p[r[h]],p[r[h+1]],s[i]->angle)<s[i]->dist+EPS) h++;
        r[e++]=s[i]-p;
    }
    while(e-h>2&&dist(p[r[e-2]],p[r[e-1]],p[r[h]].angle)<p[r[h]].dist+EPS) e--;
    while(e-h>2&&dist(p[r[h]],p[r[h+1]],p[r[e-1]].angle)<p[r[e-1]].dist+EPS) h++;
    for(int i=h;i<e;i++) r[i-h]=r[i];
    r[e-h]=r[0];
    double x=0.0,y=0.0;
    for(int i=0;i<e-h;i++)
    {
        x+=(p[r[i]].dist*sin(p[r[i+1]].angle)-p[r[i+1]].dist*sin(p[r[i]].angle))
            /sin(p[r[i+1]].angle-p[r[i]].angle);
        y+=(p[r[i+1]].dist*cos(p[r[i]].angle)-p[r[i]].dist*cos(p[r[i+1]].angle))
            /sin(p[r[i+1]].angle-p[r[i]].angle);
    }
    x/=e-h;
    y/=e-h;
    for(int i=0;i<n;i++)
        if(x*cos(p[i].angle)+y*sin(p[i].angle)<p[i].dist+EPS) return 0;
    return e-h;
}
```

4.7 Circle Polygon Intersection

```
//Circle's center lies in (0,0), polygon is given counterclockwise
#define x(_t) (xa+(_t)*a)
#define y(_t) (ya+(_t)*b)
double radian(double xa,double ya,double xb,double yb)
{
    return atan2(xa*yb-xb*ya,xa*xb+ya*yb);
}
double part(double xa,double ya,double xb,double yb,double r)
{
    double l=sqrt((xa-xb)*(xa-xb)+(ya-yb)*(ya-yb));
    double a=(xb-xa)/l,b=(yb-ya)/l,c=a*xa+b*ya;
    double d=4.0*(c*c-xa*xa-ya*ya+r*r);
    if(d<EPS) return radian(xa,ya,xb,yb)*r*r*0.5;
    else
    {
        d=sqrt(d)*0.5;
        double s=-c-d,t=-c+d;
        if(s<0.0) s=0.0; else if(s>l) s=l;
        if(t<0.0) t=0.0; else if(t>l) t=l;
```

```
        return (x(s)*y(t)-x(t)*y(s)
            +(radian(xa,ya,x(s),y(s))+radian(x(t),y(t),xb,yb))*r*r)*0.5;
    }
}
double intersection(double xp[],double yp[],int n,int r)
{
    double s=0.0;
    xp[n]=xp[0];
    yp[n]=yp[0];
    for(int i=0;i<n;i++) s+=part(xp[i],yp[i],xp[i+1],yp[i+1],r);
    return fabs(s);
}
```

4.8 Union of Rectangles

```
struct node
{
    int v,c,d;
    int m; //Length of current sweep line
    int n; //Number of borders
    bool l,r;
};
void ST_Build(int &r,int a,int b,node t[],int &tn)
{
    r=tn++;
    if(a==b) t[r].c=t[r].d=-1;
    else
    {
        int m=(a+b)>>1;
        ST_Build(t[r].c,a,m,t,tn);
        ST_Build(t[r].d,m+1,b,t,tn);
    }
}
void ST_UpdateRange(int r,int a,int b,node t[],int x,int y,int delta)
{
    if(x>y) return;
    if(a==x&&b==y) t[r].v+=delta;
    else
    {
        int m=(a+b)>>1;
        ST_UpdateRange(t[r].c,a,m,t,x,min(m,y),delta);
        ST_UpdateRange(t[r].d,m+1,b,t,max(m+1,x),y,delta);
    }
    if(t[r].v)
    {
        t[r].m=b-a+1; //*
        t[r].n=2;
        t[r].l=t[r].r=true;
    }
    else if(a==b)
    {

        t[r].m=0;
        t[r].n=0;
        t[r].l=t[r].r=false;
    }
    else
    {
        int c=t[r].c,d=t[r].d;
        t[r].m=t[c].m+t[d].m;
        t[r].n=t[c].n+t[d].n;
        t[r].l=t[c].l;
        t[r].r=t[d].r;
        if(t[c].r&&t[d].l) t[r].n-=2;
    }
}
```

4.9 Triangle Centers

```
bool Circumcenter(vect &a,vect &b,vect &c,vect &r)
{
    double d=(a.x*(b.y-c.y)+b.x*(c.y-a.y)+c.x*(a.y-b.y))*2.0;
    if(fabs(d)<EPS) return false;
```

```
    r.x=((a.x*a.x+a.y*a.y)*(b.y-c.y)+(b.x*b.x+b.y*b.y)*(c.y-a.y)
        +(c.x*c.x+c.y*c.y)*(a.y-b.y))/d;
    r.y=-((a.x*a.x+a.y*a.y)*(b.x-c.x)+(b.x*b.x+b.y*b.y)*(c.x-a.x)
        +(c.x*c.x+c.y*c.y)*(a.x-b.x))/d;
    return true;
}
double Incenter(vect &a,vect &b,vect &c,vect &r)
{
    double u=(b-c).length(),v=(c-a).length(),w=(a-b).length(),s=u+v+w;
    if(s<EPS) {r=a;return 0.0;}
    r.x=(a.x*u+b.x*v+c.x*w)/s;
    r.y=(a.y*u+b.y*v+c.y*w)/s;
    return sqrt((v+w-u)*(w+u-v)*(u+v-w)/s)*0.5;
}
bool Orthocenter(vect &a,vect &b,vect &c,vect &r)
{
    double d=a.x*(b.y-c.y)+b.x*(c.y-a.y)+c.x*(a.y-b.y);
    if(fabs(d)<EPS) return false;
    r.x=((c.x*b.x+c.y*b.y)*(c.y-b.y)+(a.x*c.x+a.y*c.y)*(a.y-c.y)
        +(b.x*a.x+b.y*a.y)*(b.y-a.y))/d;
    r.y=-((c.x*b.x+c.y*b.y)*(c.x-b.x)+(a.x*c.x+a.y*c.y)*(a.x-c.x)
        +(b.x*a.x+b.y*a.y)*(b.x-a.x))/d;
    return true;
}
void Centroid(vect p[],int n,vect &r)
{
    double s=0.0,t;
    r.x=r.y=0.0;
    p[n]=p[0];
    for(int i=0;i<n;i++,s+=t)
    {
        t=p[i].x*p[i+1].y-p[i+1].x*p[i].y;
        r.x+=t*(p[i].x+p[i+1].x);
        r.y+=t*(p[i].y+p[i+1].y);
    }
    r.x/=s*3.0;
    r.y/=s*3.0;
}
```

4.10 Delaunay Triangulation

```
#define add_edge(_u,_v) \
{ \
    int p1=avail[m++],p2=avail[m++]; \
    if(g[_u]>=0) \
    { \
        e[p1].next=e[g[_u]].next; \
        e[e[p1].next].prev=p1; \
        e[p1].prev=g[_u]; \
        e[g[_u]].next=p1; \
    } \
    else e[p1].next=e[p1].prev=p1; \
    if(g[_v]>=0) \
    { \
        e[p2].prev=e[g[_v]].prev; \
        e[e[p2].prev].next=p2; \
        e[p2].next=g[_v]; \
        e[g[_v]].prev=p2; \
    } \
    else e[p2].next=e[p2].prev=p2; \
    e[p1].v=_v;e[p2].v=_u; \
    g[_u]=e[p2].rev=p1;g[_v]=e[p1].rev=p2; \
}
#define del_edge(_i) \
{ \
    int _j=e[_i].rev,_u=e[_j].v,_v=e[_i].v; \
    avail[--m]=_i;avail[--m]=_j; \
    if(e[_i].next==_i) g[_u]=-1; \
    else \
    { \
        e[e[_i].next].prev=e[_i].prev; \
        e[e[_i].prev].next=e[_i].next; \
```

```
        if(g[_u]==_i) g[_u]=e[_i].next; \
    } \
    if(e[_j].next==_j) g[_v]=-1; \
    else \
    { \
        e[e[_j].next].prev=e[_j].prev; \
        e[e[_j].prev].next=e[_j].next; \
        if(g[_v]==_j) g[_v]=e[_j].next; \
    } \
}
struct edge
{
    int rev,v,prev,next;
};
bool Del_cmp(const vect &a,const vect &b)
{
    return a.x+EPS<b.x||a.x-EPS<b.x&&a.y<b.y;
}
void Tangent(vect p[],int c,int g[],edge e[],int &s,int &t)
{
    bool flag=true;
    s=c;
    t=c+1;
    while(flag)
    {
        flag=false;
        while(g[s]>=0)
        {
            double z=-INF;
            int d=-1,i=g[s];
            do
            {
                double a=(p[e[i].v]-p[s])/(p[t]-p[s]);
                if(a>EPS)
                {
                    double b=(p[e[i].v]-p[s])*(p[t]-p[s])/a;
                    if(z<b) {z=b;d=i;}
                }
                i=e[i].next;
            } while(i!=g[s]);
            if(d==-1) break;
            s=e[d].v;
            flag=true;
        }
        while(g[t]>=0)
        {
            double z=-INF;
            int d=-1,i=g[t];
            do
            {
                double a=(p[s]-p[t])/(p[e[i].v]-p[t]);
                if(a>EPS)
                {
                    double b=(p[s]-p[t])*(p[e[i].v]-p[t])/a;
                    if(z<b) {z=b;d=i;}
                }
                i=e[i].next;
            } while(i!=g[t]);
            if(d==-1) break;
            t=e[d].v;
            flag=true;
        }
    }
}
//Test if d is in the circle determined by a, b and c (counterclockwise)
bool InCircle(vect &a,vect &b,vect &c,vect &d)
{
    return ((d.y-a.y)*(b.x-c.x)+(d.x-a.x)*(b.y-c.y))
          *((d.x-c.x)*(b.x-a.x)-(d.y-c.y)*(b.y-a.y))
          >((d.y-c.y)*(b.x-a.x)+(d.x-c.x)*(b.y-a.y))
          *((d.x-a.x)*(b.x-c.x)-(d.y-a.y)*(b.y-c.y))+EPS;
}
```

```
void DivConq(vect p[],int a,int b,int g[],edge e[],int avail[],int &m)
{
    if(b-a<=0) return;
    if(b-a==1) {add_edge(a,b);return;}
    int c=(a+b)>>1,s,t;
    DivConq(p,a,c,g,e,avail,m);
    DivConq(p,c+1,b,g,e,avail,m);
    Tangent(p,c,g,e,s,t);
    if(g[s]>=0)
    {
        int i=g[s],d=-1;
        double z=-INF;
        do
        {
            double x=radian(p[t]-p[s],p[e[i].v]-p[s]);
            if(z<x) {z=x;d=i;}
            i=e[i].next;
        } while(i!=g[s]);
        g[s]=d;
    }
    if(g[t]>=0)
    {
        int i=g[t],d=-1;
        double z=-INF;
        do
        {
            double x=radian(p[e[i].v]-p[t],p[s]-p[t]);
            if(z<x) {z=x;d=i;}
            i=e[i].next;
        } while(i!=g[t]);
        g[t]=d;
    }
    while(true)
    {
        int sc=-1,tc=-1;
        if(g[s]>=0)
        {
            int i=e[g[s]].next;
            if((p[t]-p[s])/(p[e[i].v]-p[s])>EPS)
            {
                while(InCircle(p[s],p[t],p[e[i].v],p[e[e[i].next].v]))
                {
                    del_edge(i);
                    i=e[i].next;
                }
                sc=e[i].v;
                g[sc]=e[i].rev;
            }
        }
        if(g[t]>=0)
        {
            int i=e[g[t]].prev;
            if((p[e[i].v]-p[t])/(p[s]-p[t])>EPS)
            {
                while(InCircle(p[s],p[t],p[e[i].v],p[e[e[i].prev].v]))
                {
                    del_edge(i);
                    i=e[i].prev;
                }
                tc=e[i].v;
                g[tc]=e[i].rev;
            }
        }
        add_edge(s,t);
        if(sc==-1&&tc==-1) break;
        if(sc>=0&&(tc==-1||InCircle(p[s],p[t],p[tc],p[sc]))) s=sc; else t=tc;
    }
}
//Return number of edges. Edges for each vertex are listed counterclockwise.
//p[] will be changed. Size of e[] is larger than n*6.
int Delaunay(vect p[],int n,int g[],edge e[])
{
```

```
    sort(p,p+n,Del_cmp);
    memset(g,255,sizeof(*g)*n);
    static int avail[maxn*6];
    for(int i=0;i<n*6;i++) avail[i]=i;
    int m=0;
    DivConq(p,0,n-1,g,e,avail,m);
    return m/2;
}
```

4.11 Voronoi Diagram

```
//Using Delaunay Triangulation
struct vedge
{
    vect a,b;
    bool ai,bi; //false=INF
    int v,prev,next;
};
bool Vor_intersect(vect &a,vect &b,vect &c,vect &d)
{
    if((a-c)/(b-c)>EPS)
    {
        double s,t;
        vect p=(a+c)*0.5,q=(b+c)*0.5,u(c.y-a.y,a.x-c.x),v(c.y-b.y,b.x-c.x);
        intersect(p,q,u,v,s,t);
        d=p+u*s;
        return true;
    }
    return false;
}
int Voronoi(vect p[],int n,int g[],vedge e[])
{
    static edge f[maxn*6];
    int m=Delaunay(p,n,g,f);
    for(int i=0;i<n;i++)
        if(g[i]>=0)
        {
            int j=g[i];
            do
            {
                if(!(e[j].ai=Vor_intersect(p[f[j].v],p[f[f[j].next].v],p[i],e[j].a)))
                    e[j].a=(p[f[j].v]+p[i])*0.5;
                if(!(e[j].bi=Vor_intersect(p[f[f[j].prev].v],p[f[j].v],p[i],e[j].b)))
                    e[j].b=(p[f[j].v]+p[i])*0.5;
                if(!e[j].ai&&!e[j].bi)
                {
                    e[j].a=e[j].a+vect(p[i].y-p[f[j].v].y,p[f[j].v].x-p[i].x);
                    e[j].b=e[j].b-vect(p[i].y-p[f[j].v].y,p[f[j].v].x-p[i].x);
                }
                if((e[j].a-p[i])/(e[j].b-p[i])>EPS)
                {
                    if(e[j].bi) e[j].a=e[j].b*2.0-e[j].a;
                    else e[j].b=e[j].a*2.0-e[j].b;
                }
                e[j].v=f[j].v;
                e[j].prev=f[j].prev;
                e[j].next=f[j].next;
                j=e[j].next;
            } while(j!=g[i]);
        }
    return m;
}
```

4.12 Smallest Enclosing Circle

```
//Caculate Convex-hull first
double MinCircle(vect p[],int n,vect &q)
{
    if(n==1) {q=p[0];return 0.0;}
    for(int i=0;i<n;i++) swap(p[rand()%n],p[rand()%n]);
    double r=(p[0]-p[1])*(p[0]-p[1])/4.0;
    q=(p[0]+p[1])*0.5;
    for(int i=2;i<n;i++)
```

```
    {
        if((p[i]-q)*(p[i]-q)<r+EPS) continue;
        r=(p[i]-p[0])*(p[i]-p[0])/4.0;
        q=(p[i]+p[0])*0.5;
        for(int j=1;j<i;j++)
        {
            if((p[j]-q)*(p[j]-q)<r+EPS) continue;
            r=(p[i]-p[j])*(p[i]-p[j])/4.0;
            q=(p[i]+p[j])*0.5;
            for(int k=0;k<j;k++)
            {
                if((p[k]-q)*(p[k]-q)<r+EPS) continue;
                Circumcenter(p[i],p[j],p[k],q);
                r=(p[i]-q)*(p[i]-q)/4.0;
            }
        }
    }
    return sqrt(r);
}
```

5 General Data Structures and Algorithms

5.1 Quick Sort

```
void qsort(int s[],int a,int b)
{
    if(a>=b) return;
    int m=s[(a+b)>>1],i=a-1,j=b+1;
    while(true)
    {
        while(s[++i]<m);
        while(s[--j]>m);
        if(i>=j) break;
        swap(s[i],s[j]);
    }
    qsort(s,a,i-1);
    qsort(s,j+1,b);
}
```

5.2 Merge Sort

```
int msort(int s[],int a,int b)
{
    if(a>=b) return 0;
    int m=(a+b)>>1,r=msort(s,a,m)+msort(s,m+1,b),i=a,j=m+1,k=a;
    static int t[maxn];
    for(;k<=b;k++)
        if(j>b||(i<=m&&s[i]<=s[j])) {t[k]=s[i++];r+=j-m-1;}
        else t[k]=s[j++];
    memcpy(s+a,t+a,sizeof(*s)*(b-a+1));
    return r;
}
```

5.3 Quick Select

```
int qselect(int s[],int a,int b,int k) //k in [1,n]
{
    if(a==b) return s[a];
    int m=s[(a+b)>>1],i=a,j=b;
    while(true)
    {
        while(m>s[i]) i++;
        while(m<s[j]) j--;
        if(i>=j) break;
        swap(s[i++],s[j--]);
    }
    if(i==j) i++;
    return k<=i?qselect(s,a,j,k):qselect(s,i,b,k);
}
```

5.4 KMP

```
int KMP(char s[],char t[],int n,int m)
{
    int r[maxn];
    for(int i=1,j=r[0]=-1;i<m;i++)
    {
        while(j>=0&&t[i-1]!=t[j]) j=r[j];
        r[i]=++j;
    }
    for(int i=0,j=0;i<n;i++,j++)
    {
        while(j>=0&&s[i]!=t[j]) j=r[j];
        if(j==m-1) return i-m+1;
    }
    return -1;
}
```

5.5 RMQ – Sparse Table

```
void RMQ_Init(int s[],int n)
{
    for(int i=1,j=0,k=n;i+i<=n;i<<=1)
    {
        for(int w=k-i;j<w;j++,k++) s[k]=max(s[j],s[j+i]); //*
        j+=i;
    }
}
int RMQ_Query(int s[],int n,int a,int b)
{
    int k=0;
    for(;(1<<(k+1))<=b-a;k++);
    int w=k*(n+1)-(1<<k)+1;
    return max(s[w+a],s[w+1+b-(1<<k)]); //*
}
void RMQ2_Init(int s[][1<<logn][logn+1][logn+1],int n,int m)
{
    for(int u=0;(1<<u)<=n;u++)
        for(int v=0;(1<<v)<=m;v++)
        {
            if(u==0&&v==0) continue;
            int p=1<<u,q=1<<v;
            for(int i=0;i+p<=n;i++)
                for(int j=0;j+q<=m;j++)
                    if(u>0)
                        s[i][j][u][v]=max(s[i][j][u-1][v],s[i+(p>>1)][j][u-1][v]); //*
                    else
                        s[i][j][u][v]=max(s[i][j][u][v-1],s[i][j+(q>>1)][u][v-1]); //*
        }
}
int RMQ2_Query(int s[][1<<logn][logn+1][logn+1],int n,int m,int i,int j,int p,int q)
{
    int u=0,v=0;
    while((1<<(u+1))<=p-i) u++;
    while((1<<(v+1))<=q-j) v++;
    return max( max(s[i][j][u][v],s[p-(1<<u)+1][q-(1<<v)+1][u][v]),
                max(s[p-(1<<u)+1][j][u][v],s[i][q-(1<<v)+1][u][v])); //*
}
```

5.6 Binary Heap

```
//Use priority queue when possible.
//h[1] is the index of minimum element. Initialize h[] and r[] manually.
void BH_Up(int h[],int r[],int d[],int p)
{
    int i=r[p];
    for(;i>1&&d[p] < d[h[i>>1]];i>>=1) //*
        {h[i]=h[i>>1];r[h[i]]=i;}
    h[i]=p;r[p]=i;
}
void BH_Down(int h[],int r[],int d[],int n,int p)
{
    for(int i=r[p];;)
```

```
    {
        int j=i<<1;
        if(j<n&&d[h[j+1]] < d[p]&&d[h[j+1]] < d[h[j]]) //*
            {h[i]=h[j+1];r[h[i]]=i;i=j+1;}
        else if(j<=n&&d[h[j]] < d[p]) //*
            {h[i]=h[j];r[h[i]]=i;i=j;}
        else {h[i]=p;r[p]=i;break;}
    }
}
void BH_Push(int h[],int r[],int d[],int &n,int p)
{
    h[++n]=p;r[p]=n;
    BH_Up(h,r,d,p);
}
int BH_Pop(int h[],int r[],int d[],int &n)
{
    int t=h[1];
    h[1]=h[n--];r[h[1]]=1;
    BH_Down(h,r,d,n,h[1]);
    return t;
}
void BH_Build(int h[],int r[],int d[],int n)
{
    for(int i=n>>1;i>0;i--) BH_Down(h,r,d,n,h[i]);
}
```

5.7 Segment Tree

```
//* : Require modify for RMQ. Only covers range [0,n)
void ST_Build(int s[],int n)
{
    int a=0,b=n,j,m;
    while(n>1)
    {
        m=(n+1)>>1;
        for(int i=0;i<m;i++)
            if((j=i<<1)+1==n) s[i+b]=s[j+a];
            else s[i+b]=s[j+a]+s[j+a+1]; //*
        a=b;b+=m;n=m;
    }
}
void ST_Update(int s[],int n,int j,int v)
{
    int a=0,b=n,i,m;
    s[j]=v;
    while(n>1)
    {
        m=(n+1)>>1;
        i=j>>1;j=i<<1;
        if(j+1==n) s[i+b]=s[j+a];
        else s[i+b]=s[j+a]+s[j+a+1]; //*
        a=b;b+=m;n=m;j=i;
    }
}
int ST_Query(int s[],int n,int x,int y)
{
    int a=0,c=0; //*
    while(x<=y)
    {
        if(x&1) c+=s[a+x++]; //*
        if(!(y&1)) c+=s[a+y--]; //*
        x>>=1;y>>=1;a+=n;n=(n+1)>>1;
    }
    return c;
}
//Below functions conflict with above.
void ST_Update(int s[],int n,int x,int y,int delta)
{
    int a=0;
    while(x<=y)
    {
        if(x&1) s[a+x++]+=delta;
```

```
        if(!(y&1)) s[a+y--]+=delta;
        x>>=1;y>>=1;a+=n;n=(n+1)>>1;
    }
}
int ST_Query(int s[],int n,int i)
{
    int a=0,c=0;
    while(n>1) {c+=s[i+a];i>>=1;a+=n;n=(n+1)>>1;}
    return c+s[a];
}

//An lucid version:
struct node
{
    int v,c,d;
};
void ST_Build(int &r,int a,int b,node t[],int &tn,int *s)
{
    r=tn++;
    if(a==b)
    {
        t[r].c=t[r].d=-1;
        t[r].v=s[a];
    }
    else
    {
        int m=(a+b)>>1;
        ST_Build(t[r].c,a,m,t,tn,s);
        ST_Build(t[r].d,m+1,b,t,tn,s);
        t[r].v=t[t[r].c].v+t[t[r].d].v; //*
    }
}
void ST_Update(int r,int a,int b,node t[],int p,int v)
{
    if(a==b) t[r].v=v;
    else
    {
        int m=(a+b)>>1;
        if(p<=m) ST_Update(t[r].c,a,m,t,p,v); else ST_Update(t[r].d,m+1,b,t,p,v);
        t[r].v=t[t[r].c].v+t[t[r].d].v; //*
    }
}
int ST_Query(int r,int a,int b,node t[],int x,int y)
{
    if(x<=a&&y>=b) return t[r].v;
    else if(x>y) return 0; //*
    else
    {
        int m=(a+b)>>1;
        return ST_Query(t[r].c,a,m,t,x,min(m,y))
        +ST_Query(t[r].d,m+1,b,t,max(m+1,x),y); //*
    }
}
void ST_UpdateRange(int r,int a,int b,node t[],int x,int y,int delta)
{
    if(x>y) return;
    t[r].v+=(y-x+1)*delta;
    if(x>a||y<b)
    {
        int m=(a+b)>>1;
        ST_UpdateRange(t[r].c,a,m,t,x,min(m,y),delta);
        ST_UpdateRange(t[r].d,m+1,b,t,max(m+1,x),y,delta);
    }
}
```

5.8 Binary Indexed Tree 2D

```
//Range: t[1..m][1..n], u in [1..m], v in [1..n]
#define lowbit(_x) (_x&(_x^(_x-1)))
void BIT_Update(int t[][maxn],int m,int n,int u,int v,int delta)
{
    int a=v;
```

```
    while(u<=m)
    {
        v=a;
        while(v<=n) {t[u][v]+=delta;v+=lowbit(v);}
        u+=lowbit(u);
    }
}
//Return sum[1..u][1..v]
int BIT_Query(int t[][maxn],int m,int n,int u,int v)
{
    int a=v,s=0;
    while(u>0)
    {
        v=a;
        while(v>0) {s+=t[u][v];v-=lowbit(v);}
        u-=lowbit(u);
    }
    return s;
}
```

5.9 Trie

```
struct node {char a;int b,c,n;};
//Set r=-1 for the first call to Insert
int Insert(int *r,char *s,node t[],int &m)
{
    if(!*s) return 0;
    while(true)
    {
        while(*r!=-1&&t[*r].a!=*s) r=&t[*r].b;
        if(*r==-1) {t[m].a=*s;t[m].b=t[m].c=-1;t[m].n=0;*r=m++;}
        if(*++s) r=&(t[*r].c); else return ++t[*r].n;
    }
}
//Return: -1=Not found, 0=Not found but preffix, >0=Found.
int Match(int r,char *s,node *t)
{
    if(!*s) return -1;
    while(true)
    {
        while(r!=-1&&t[r].a!=*s) r=t[r].b;
        if(r==-1) return -1;
        if(*++s) r=t[r].c; else return t[r].n;
    }
}
```

5.10 Suffix Array

```
//Add '\1' between each string. The whole sequence ends with '\0'.
void SA_Sort(char s[],int n,int list[],int rank[])
{
    int c[256]={0};
    for(int i=0;i<n;i++) c[(int)s[i]]++;
    for(int i=1;i<256;i++) c[i]+=c[i-1];
    for(int i=0;i<n;i++) list[--c[(int)s[i]]]=i;
    rank[list[0]]=0;
    for(int i=1;i<n;i++)
    {
        rank[list[i]]=rank[list[i-1]];
        if(s[list[i]]!=s[list[i-1]]) rank[list[i]]++;
    }
    for(int w=1;w<n;w<<=1)
    {
        int h[maxn],x[maxn],r[maxn];
        memset(h,255,sizeof(*h)*n);
        memcpy(r,rank,sizeof(*r)*n);
        for(int i=n-1;i>=0;i--)
        {
            int j=list[i]-w;
            if(j<0) j+=n;
            x[j]=h[r[j]];
            h[r[j]]=j;
        }
```

```
        for(int i=0,j=0;i<n;i++)
            for(int k=h[i];k!=-1;k=x[k]) list[j++]=k;
        rank[list[0]]=0;
        for(int i=1;i<n;i++)
        {
            rank[list[i]]=rank[list[i-1]];
            if(r[list[i]]!=r[list[i-1]]||r[list[i]+w]!=r[list[i-1]+w])
                rank[list[i]]++;
        }
    }
}
//h[i]=LCP(Suffix(list[i]),Suffix(list[i-1]))
void SA_Height(char s[],int n,int list[],int rank[],int h[])
{
    h[0]=0;
    for(int i=0,k=0;i<n;i++)
        if(rank[i])
        {
            while(s[i+k]==s[list[rank[i]-1]+k]) k++;
            h[rank[i]]=k;
            if(k) k--;
        }
        else k=0;
}
```

5.11 Union-find

```
void MakeSet(int f[],int a)
{
    f[a]=a;
}
int Find(int f[],int a)
{
    if(f[a]==a) return a;
    return f[a]=Find(f,f[a]);
}
bool Same(int f[],int a,int b)
{
    return Find(f,a)==Find(f,b);
}
void Union(int f[],int a,int b)
{
    f[Find(f,a)]=Find(f,b);
}
```

5.12 Splay Tree

```
struct node {int l,r,p,w;};
void ST_RotateL(node t[],int x)
{
    int y=t[x].p,z=t[y].p;
    t[x].p=z;
    if(z!=-1)
    {
        if(t[z].l==y) t[z].l=x; else t[z].r=x;
    }
    t[y].r=t[x].l;
    if(t[y].r!=-1) t[t[y].r].p=y;
    t[x].l=y;
    t[y].p=x;
}
void ST_RotateR(node t[],int x)
{
    int y=t[x].p,z=t[y].p;
    t[x].p=z;
    if(z!=-1)
    {
        if(t[z].l==y) t[z].l=x; else t[z].r=x;
    }
    t[y].l=t[x].r;
    if(t[y].l!=-1) t[t[y].l].p=y;
    t[x].r=y;
    t[y].p=x;
```

```
}
int ST_Splay(node t[],int x)
{
    while(t[x].p!=-1)
    {
        int y=t[x].p,z=t[y].p;
        if(z==-1)
        {
            if(t[y].l==x) ST_RotateR(t,x); else ST_RotateL(t,x);
        }
        else if(t[z].l==y)
        {
            if(t[y].l==x) {ST_RotateR(t,y);ST_RotateR(t,x);}
            else {ST_RotateL(t,x);ST_RotateR(t,x);}
        }
        else
        {
            if(t[y].r==x) {ST_RotateL(t,y);ST_RotateL(t,x);}
            else {ST_RotateR(t,x);ST_RotateL(t,x);}
        }
    }
    return x;
}
int ST_Find(node t[],int r,int w)
{
    while(r!=-1&&t[r].w!=w)
    {
        if(w<t[r].w) r=t[r].l; else r=t[r].r;
    }
    if(r==-1) return -1;
    return ST_Splay(t,r);
}
int ST_Insert(node t[],int r,int x)
{
    int p=-1;
    while(r!=-1)
    {
        p=r;
        if(t[x].w<t[r].w) r=t[r].l; else r=t[r].r;
    }
    if(p==-1) return x;
    if(t[x].w<t[p].w) t[p].l=x; else t[p].r=x;
    t[x].p=p;
    return ST_Splay(t,x);
}
int ST_Delete(node t[],int x)
{
    while(t[x].l!=-1||t[x].r!=-1)
    {
        if(t[x].l!=-1) ST_RotateR(t,t[x].l);
        if(t[x].r!=-1) ST_RotateL(t,t[x].r);
    }
    int y=t[x].p;
    if(y!=-1)
    {
        if(t[y].l==x) t[y].l=-1; else t[y].r=-1;
        t[x].p=-1;
    }
    return ST_Splay(t,y);
}
```

5.13 Gauss Elimination

```
//Return: x[]==Solutions, -1==No solution, 0=OK, 1=More than one solution
int Gauss(double a[][maxn],int n,int m,double x[])
{
    const double EPS=1e-10;
    int p[maxn],q=-1;
    double r[maxn];
    for(int i=0;i<n;i++)
    {
        p[i]=-1;
```

```
        for(int j=q+1;j<m&&p[i]==-1;j++)
            for(int k=i;k<n;k++)
                if(fabs(a[k][j])>EPS)
                {
                    memcpy(r,a[k],(m+1)*sizeof(double));
                    if(k>i)
                    {
                        memcpy(a[k],a[i],(m+1)*sizeof(double));
                        memcpy(a[i],r,(m+1)*sizeof(double));
                    }
                    p[i]=q=j;
                    break;
                }
        if(p[i]==-1)
        {
            for(int j=i;j<n;j++)
                if(fabs(a[j][m])>EPS) return -1;
            n=i;
            break;
        }
        for(int j=i+1;j<n;j++)
            if(fabs(a[j][q])>EPS)
            {
                double c=a[j][q]/r[q];
                a[j][q]=0.0;
                for(int k=q+1;k<=m;k++) a[j][k]-=c*r[k];
            }
    }
    if(m==n)
    {
        for(int i=n-1;i>=0;i--)
        {
            for(int j=i+1;j<n;j++)
            {
                a[i][m]-=a[j][m]*a[i][j]/a[j][j];
                a[i][j]=0.0;
            }
            x[i]=a[i][m]/a[i][i];
        }
        return 0;
    }
    memset(x,0,m*sizeof(double));
    for(int i=n-1;i>=0;i--)
    {
        for(int j=i+1;j<n;j++)
        {
            double c=a[i][p[j]]/a[j][p[j]];
            for(int k=p[j]+1;k<=m;k++) a[i][k]-=c*a[j][k];
            a[i][p[j]]=0.0;
        }
        x[p[i]]=a[i][m]/a[i][p[i]];
    }
    return 1;
}
```

5.14 Fast Fourier Transform

```
//y=a+ib
void FFT(int &n,double a[],double b[],bool inverse)
{
    int t;
    for(t=1;((t-1)&(n-1))!=n-1;t<<=1);
    memset(a+n,0,(t-n)*sizeof(*a));
    memset(b+n,0,(t-n)*sizeof(*b));
    n=t;
    for(int i=0;i<n;i++)
    {
        t=0;
        for(int j=n,k=i;j>1;j>>=1,k>>=1) t=(t<<1)|(k&1);
        if(t<=i) continue;
        swap(a[i],a[t]);
        swap(b[i],b[t]);
```

```
    }
    for(int i=1;i<n;i<<=1)
        for(int j=0;j<n;j+=i<<1)
            for(int k=0;k<i;k++)
            {
                double c=cos(PI/i*k),d=sin(PI/i*k);
                if(inverse) d=-d;
                double e=c*a[i+j+k]-d*b[i+j+k],f=c*b[i+j+k]+d*a[i+j+k];
                a[i+j+k]=a[j+k]-e;
                b[i+j+k]=b[j+k]-f;
                a[j+k]+=e;
                b[j+k]+=f;
            }
    if(inverse) for(int i=0;i<n;i++) {a[i]/=n;b[i]/=n;}
}
```

5.15 Linear Programming – Simplex

```
//Linear Programming - Simplex
//Input: y[]: base vector, where y[] form an identity matrix, a.x=b>=0
//Return: -1: Infinity, 0: OK, 1: More than one solution
//Output: x[y[i]]=b[i], which maximum c.x
int Simplex(int n,int m,double a[][maxn],double b[],double c[],int y[])
{
    bool u[maxn]={};
    for(int i=0;i<n;i++) u[y[i]]=true;
    while(true)
    {
        int p,q=-1,ret=0;
        for(p=0;p<m;p++)
            if(!u[p])
            {
                double t=c[p];
                for(int i=0;i<n;i++) t-=a[i][p]*c[y[i]];
                if(fabs(t)<EPS) ret=1;
                if(t>0.0) break;
            }
        if(p==m) return ret;
        double z=INF;
        for(int i=0;i<n;i++)
            if(a[i][p]>EPS&&z>b[i]/a[i][p]) {q=i;z=b[i]/a[i][p];}
        if(q<0) return -1;
        u[y[q]]=false;
        u[p]=true;
        y[q]=p;
        double r=a[q][p];
        b[q]/=r;
        for(int i=0;i<m;i++) a[q][i]/=r;
        for(int i=0;i<n;i++)
            if(i!=q&&fabs(a[i][p])>EPS)
            {
                r=a[i][p];
                b[i]-=r*b[q];
                for(int j=0;j<m;j++) a[i][j]-=r*a[q][j];
            }
    }
}
```

5.16 Aho-Corasick Automation

```
const int sigma=26; //Size of alphabet
const char symbol='a'; //First letter
struct aca_node
{
    int next[sigma],prev,id;
};
int ACA_Insert(aca_node aca[],int &m,char *str,int &d)
{
    int r=0;
    for(;*str;str++)
    {
        int key=*str-symbol;
        if(aca[r].next[key]<0)
```

```
        {
            memset(&aca[m],255,sizeof(aca[m]));
            aca[r].next[key]=m++;
        }
        r=aca[r].next[key];
    }
    if(aca[r].id<0) aca[r].id=d++;
    return aca[r].id;
}
void ACA_Init(aca_node aca[])
{
    int m=1,d=0;
    memset(&aca[0],255,sizeof(aca[0]));
    /* id=ACA_Insert(aca,m,str,d); */
    queue<int> q;
    q.push(0);
    while(!q.empty())
    {
        int r=q.front();
        q.pop();
        for(int i=0;i<sigma;i++)
        {
            int t=aca[r].next[i];
            if(t>=0)
            {
                int p=aca[r].prev;
                while(p>=0&&aca[p].next[i]<0) p=aca[p].prev;
                aca[t].prev=p<0?0:aca[p].next[i];
                q.push(t);
            }
        }
    }
}
void ACA_Match(aca_node aca[],char *str)
{
    for(int r=0;*str;str++)
    {
        int key=*str-symbol;
        while(r>=0&&aca[r].next[key]<0) r=aca[r].prev;
        if(r>=0)
        {
            r=aca[r].next[key];
            int p=r;
            while(p>=0)
            {
                /* Matched pattern's id == aca[p].id */
                p=aca[p].prev;
            }
        }
        else r=0;
    }
}
```

6 Classic Problems

6.1 2-satisfiability

```
//Using Strongly Connected Component
//Vertex: 0..n-1=true, n..2n-1=false
bool sat(int g[],int f[],edge e[],int n,bool s[])
{
    static int r[maxn],q[maxn],c[maxn],l[maxn],a[maxn];
    int h=0,d=0,m=StrongCC(g,f,e,n*2,r);
    for(int i=0;i<n;i++) if(r[i]==r[i+n]) return false;
    memset(c,0,sizeof(*c)*m);
    memset(a,255,sizeof(*a)*m);
    for(int i=0;i<n*2;i++)
    {
        l[i]=a[r[i]];
        a[r[i]]=i;
        for(int j=g[i];j!=-1;j=e[j].next)
```

```
            if(r[i]!=r[e[j].v]) c[r[e[j].v]]++;
    }
    for(int i=0;i<m;i++) if(c[i]==0) q[d++]=i;
    while(h<d)
    {
        int t=q[h++];
        for(int i=a[t];i!=-1;i=l[i])
            for(int j=g[i];j!=-1;j=e[j].next)
            {
                int v=r[e[j].v];
                if(t==v) continue;
                if(--c[v]==0) q[d++]=v;
            }
    }
    static char u[maxn];
    memset(u,255,sizeof(u));
    for(int k=m-1;k>=0;k--)
    {
        int t=q[k];
        if(u[t]!=-1) continue;
        u[t]=1;
        for(int i=a[t];i!=-1&&u[t];i=l[i])
            for(int j=g[i];j!=-1;j=e[j].next)
                if(u[r[e[j].v]]==0) {u[t]=0;break;}
        for(int i=a[t];i!=-1;i=l[i])
            if(i<n) u[r[i+n]]=1-u[r[i]]; else u[r[i-n]]=1-u[r[i]];
    }
    for(int i=0;i<n;i++) s[i]=u[r[i]]?true:false;
    return true;
}
```

6.2 Unix Time

```
struct datetime
{
    int64_t year;
    int month,day,hour,min,sec;
};
int64_t toUnix(datetime t)
{
    static int month[13]={0,31,59,90,120,151,181,212,243,273,304,334,365};
    int64_t day=0;
    if(t.year>=1970)
    {
        day=(t.year-1970)*365+(t.year-1969)/4
            -(t.year-1701)/400-(t.year-1801)/400-(t.year-1901)/400;
        day+=month[t.month-1]+t.day-1;
        if(t.month>2&&((t.year%4==0&&t.year%100!=0)||t.year%400==0)) day++;
        return day*86400+t.hour*3600+t.min*60+t.sec;
    }
    else
    {
        day=(1969-t.year)*365+(1971-t.year)/4
            -(2299-t.year)/400-(2199-t.year)/400-(2099-t.year)/400;
        day+=month[12]-month[t.month-1]-t.day;
        if(t.month<=2&&((t.year%4==0&&t.year%100!=0)||t.year%400==0)) day++;
        return -(day*86400+(23-t.hour)*3600+(59-t.min)*60+60-t.sec);
    }
}
datetime fromUnix(int64_t d)
{
    static int month[13]={0,31,59,90,120,151,181,212,243,273,304,334,365};
    datetime t={};
    if(d>=0)
    {
        int64_t day=d/86400;
        d%=86400;
        t.year=1970+day/146097*400;
        day%=146097;
        if(day>=120895) day+=3;
        else if(day>=84371) day+=2;
        else if(day>=47847) day++;
```

```
        t.year+=day/1461*4;
        day%=1461;
        if(day>=365) {day-=365;t.year++;}
        if(day>=365) {day-=365;t.year++;}
        if(day>=366) {day-=366;t.year++;}
        if((t.year%4==0&&t.year%100!=0)||t.year%400==0)
        {
            if(day==59) {t.month=2;t.day=29;}
            else if(day>59) day--;
        }
        if(t.month==0)
            for(int i=11;i>=0;i--)
                if(day>=month[i])
                {
                    t.month=i+1;
                    t.day=day-month[i]+1;
                    break;
                }
        t.hour=d/3600;
        t.min=d/60%60;
        t.sec=d%60;
    }
    else
    {
        d=-d-1;
        int64_t day=d/86400;
        d=d%86400+1;
        t.year=1969-day/146097*400;
        day%=146097;
        if(day>98614) day+=3;
        else if(day>62090) day+=2;
        else if(day>25566) day++;
        t.year-=day/1461*4;
        day%=1461;
        if(day>=365) {day-=365;t.year--;}
        if(day>=366)
        {
            day-=366;
            t.year--;
            if(day>=365) {day-=365;t.year--;}
        }
        if((t.year%4==0&&t.year%100!=0)||t.year%400==0)
        {
            if(day==306) {t.month=2;t.day=29;}
            else if(day>306) day--;
        }
        if(t.month==0)
            for(int i=1;i<=12;i++)
                if(day>=365-month[i])
                {
                    t.month=i;
                    t.day=365-month[i-1]-day;
                    break;
                }
        t.hour=(86400-d)/3600;
        t.min=(86400-d)/60%60;
        t.sec=(86400-d)%60;
    }
    return t;
}
```

6.3 NFA to DFA

```
vector<int> eclose(vector<int> nfa[][sigma+1],int n,vector<int> p)
{
    static bool u[maxn]={};
    queue<int> q;
    for(vector<int>::iterator i=p.begin();i!=p.end();i++) {u[*i]=true;q.push(*i);}
    while(!q.empty())
    {
        int x=q.front();
        q.pop();
```

```
        for(vector<int>::iterator i=nfa[x][sigma].begin();i!=nfa[x][sigma].end();i++)
            if(!u[*i]) {q.push(*i);p.push_back(*i);u[*i]=true;}
    }
    for(vector<int>::iterator i=p.begin();i!=p.end();i++) u[*i]=false;
    sort(p.begin(),p.end());
    return p;
}
//State 0: start state
void NFAtoDFA
    (vector<int> nfa[][sigma+1],int n,int dfa[][sigma],vector<int> s[],int &m)
{
    map<vector<int>,int> hash;
    s[0]=eclose(nfa,n,vector<int>(1,0));
    hash[s[0]]=0;
    m=1;
    int h=0;
    while(h<m)
    {
        vector<int> &x=s[h++];
        for(int i=0;i<sigma;i++)
        {
            vector<int> y;
            static bool u[maxn]={};
            for(vector<int>::iterator j=x.begin();j!=x.end();j++)
                for(vector<int>::iterator k=nfa[*j][i].begin();
                    k!=nfa[*j][i].end();k++)
                    if(!u[*k]) {u[*k]=true;y.push_back(*k);}
            for(vector<int>::iterator j=y.begin();j!=y.end();j++) u[*j]=false;
            y=eclose(nfa,n,y);
            if(hash.find(y)==hash.end()) {hash[y]=m;dfa[h][i]=m;s[m++]=y;}
            else dfa[h][i]=hash[y];
        }
    }
}
```

question = to ? be : !be ;