Algorithm Library

CReatiQ

South China Normal University

June 1, 2025

Contents

数学

Set Xor-Min

维护一个集合 S，可以求 $\min_{y\in S}(x\oplus y)$。

```
struct SetXorMin
{
    static constexpr int L=30;
    int tot=0;
    vector<array<int,2>> c;
    vector<int> s;
    set<i64> in;

    SetXorMin() {}
    SetXorMin(int n)
    {
        c.resize((n+1)*(L+1));
        s.resize((n+1)*(L+1));
    }

    void insert(i64 x)
    {
        if (in.count(x))
            return;
        in.insert(x);
        int p=0;
        for (int i=L;i>=0;i--)
        {
            bool o=x>>i&1;
            if (!c[p][o])
                c[p][o]=++tot;
            s[p=c[p][o]]++;
        }
    }

    void erase(i64 x)
    {
        if (!in.count(x))
            return;
        in.erase(x);
        int p=0;
        for (int i=L;i>=0;i--)
        {
            bool o=x>>i&1;
            s[p=c[p][o]]--;
        }
    }

    i64 QueryXorMin(i64 x)
    {
        int p=0;
        i64 r=0;
        for (int i=L;i>=0;i--)
        {
            bool o=x>>i&1;
            if (s[c[p][o]])
                p=c[p][o];
            else
            {
                p=c[p][o^1];
                r|=1ll<<i;
            }
        }
        return r;
    }
};
```

数据结构

半群 deque

维护一个半群的 deque，支持前后增删及求和。

```
template <class T>
struct SWAG
{
    vector<T> l,sl,r,sr;

    void push_front(const T &o)
    {
        sl.push_back(sl.empty()?o:o+sl.back());
        l.push_back(o);
    }

    void push_back(const T &o)
    {
        sr.push_back(sr.empty()?o:sr.back()+o);
        r.push_back(o);
    }

    void pop_front()
    {
        if (!l.empty())
        {
            l.pop_back();
            sl.pop_back();
            return;
        }
        int n=r.size(),m;
        if (m=n-1>>1)
        {
            l.resize(m);
            sl.resize(m);
            for (int i=1;i<=m;i++)
                l[m-i]=r[i];
            sl[0]=l[0];
            for (int i=1;i<m;i++)
                sl[i]=l[i]+sl[i-1];
        }
        for (int i=m+1;i<n;i++)
            r[i-(m+1)]=r[i];
        m=n-(m+1);
        r.resize(m);
        sr.resize(m);
        if (m)
        {
            sr[0]=r[0];
            for (int i=1;i<m;i++)
                sr[i]=sr[i-1]+r[i];
        }
    }

    void pop_back()
    {
        if (!r.empty())
        {
            r.pop_back();
            sr.pop_back();
        }
        else
        {
            int n=l.size(),m;
            if (m=n-1>>1)
            {
                r.resize(m);
                sr.resize(m);
                for (int i=1;i<=m;i++)
                    r[m-i]=l[i];
                sr[0]=r[0];
                for (int i=1;i<m;i++)
                    sr[i]=sr[i-1]+r[i];
            }
            for (int i=m+1;i<n;i++)
                l[i-(m+1)]=l[i];
            m=n-(m+1);
            l.resize(m);
            sl.resize(m);
            if (m)
            {
                sl[0]=l[0];
                for (int i=1;i<m;i++)
                    sl[i]=l[i]+sl[i-1];
            }
        }
    }

    T ask()
    {
        assert(l.size()||r.size());
        if (l.size()&&r.size())
            return sl.back()+sr.back();
        return l.size()?sl.back():sr.back();
    }
};

struct Info
{
    Z k,b;

    Info operator + (const Info &o) const
    {
        return {k*o.k,b*o.k+o.b};
    }
};

Z operator + (const Z &x,const Info &o)
{
    return o.k*x+o.b;
}
```

区间众数

```
template <class T>
struct Mode
{
    int n,ksz,m;
    vector<T> b;
    vector<vector<int>> pos,f;
    vector<int> a,blk,id,l;

    Mode(const vector<T> &c):n(c.size()),ksz(max<int>(1,sqrt(n))),
        m((n+ksz-1)/ksz),b(c),pos(n),f(m,vector<int>(m)),a(n),blk(n),id(n),l(m+1)
    {
        sort(b.begin(),b.end());
        b.erase(unique(b.begin(),b.end()),b.end());
        for (int i=0;i<n;i++)
        {
            a[i]=lower_bound(b.begin(),b.end(),c[i])-b.begin();
            id[i]=pos[a[i]].size();
            pos[a[i]].push_back(i);
        }
        for (int i=0;i<n;i++)
            blk[i]=i/ksz;
        for (int i=0;i<=m;i++)
            l[i]=min(i*ksz,n);

        vector<int> cnt(b.size());
        for (int i=0;i<m;i++)
        {
            cnt.assign(b.size(),0);
            pair<int,int> cur={0,0};
            for (int j=i;j<m;j++)
            {
                for (int k=l[j];k<l[j+1];k++)
                    cur=max(cur,{++cnt[a[k]],a[k]});
                f[i][j]=cur.second;
            }
        }
    }

    pair<T,int> ask(int L,int R)
    {
        int val=blk[L]==blk[R-1]?0:f[blk[L]+1][blk[R-1]-1],i;
        int cnt=lower_bound(pos[val].begin(),pos[val].end(),R)-
                lower_bound(pos[val].begin(),pos[val].end(),L);
        for (int i=min(R,l[blk[L]+1])-1;i>=L;i--)
        {
            auto &v=pos[a[i]];
            while (id[i]+cnt<v.size()&&v[id[i]+cnt]<R)
                cnt++,val=a[i];
            if (a[i]>val&&id[i]+cnt-1<v.size()&&v[id[i]+cnt-1]<R)
                val=a[i];
        }
        for (int i=max(L,l[blk[R-1]]);i<R;i++)
        {
            auto &v=pos[a[i]];
            while (id[i]>=cnt&&v[id[i]-cnt]>=L)
                cnt++,val=a[i];
            if (a[i]>val&&id[i]>=cnt-1&&v[id[i]-cnt+1]>=L)
                val=a[i];
        }
        return {b[val],cnt};
    }
};
```

李超树

```
constexpr i64 inf=9e18;

template <class Info>
struct SGT
{
    int cnt=0;
    vector<Info> a;
    vector<int> ls,rs;
    i64 z,y,L,R;

    SGT(int n,i64 l,i64 r)
    {
        int N=(n+7)*64;
        a.resize(N);
        ls.resize(N);
        rs.resize(N);
        L=l,R=r,cnt=1;
        a[1]={0,inf};
    }

private:
    void insert(int &p,i64 l,i64 r,Info v)
    {
        if (!p)
        {
            p=++cnt;
            a[p]={0,inf};
        }
        i64 m=(l+r)>>1;
        if (z<=l&&r<=y)
        {
            if (a[p].y(m)>v.y(m)) swap(a[p],v);
            if (a[p].y(l)>v.y(l)) insert(ls[p],l,m,v);
            else if (a[p].y(r)>v.y(r)) insert(rs[p],m+1,r,v);
            return;
        }
        if (z<=m) insert(ls[p],l,m,v);
        if (y>m) insert(rs[p],m+1,r,v);
    }
public:
    void insert(i64 l,i64 r,const Info &v)
    {
        z=l,y=r;
        int p=1;
        insert(p,L,R,v);
    }

    i64 QueryMin(i64 p)
    {
        i64 res=a[1].y(p),l=L,r=R,x=1;
        while (l<r)
        {
            i64 m=(l+r)>>1;
            if (p<=m)
                x=ls[x],r=m;
            else
                x=rs[x],l=m+1;
            if (!x) return res;
            res=min(res,a[x].y(p));
        }
        return res;
    }
};

struct Info
{
    i64 k,b;

    i64 y(const i64 &x) const { return k*x+b; }
};
```

Splay

```
template <class Info,class Tag>
struct Splay
{
#define _rev
    struct Node
    {
        Node *c[2],*f;
        int siz;
        Info s,v;
        Tag t;

        Node():c{},f(0),siz(1),s(),v(),t() {}
        Node(Info x):c{},f(0),siz(1),s(x),v(x),t() {}

        void operator += (const Tag &o)
        {
            s+=o,v+=o,t+=o;
#ifdef _rev
            if (o.rev) swap(c[0],c[1]);
#endif
        }

        void pushup()
        {
            if (c[0])
                s=c[0]->s+v,siz=c[0]->siz+1;
            else s=v,siz=1;
            if (c[1])
                s=s+c[1]->s,siz+=c[1]->siz;
        }

        void pushdown()
        {
            for (auto x:c)
                if (x)
                    *x+=t;
            t=Tag();
        }

        void zigzag()
        {
            Node *y=f,*z=y->f;
            bool isl=y->c[0]==this;
            if (z) z->c[z->c[1]==y]=this;
            f=z,y->f=this;
            y->c[isl^1]=c[isl];
            if (c[isl]) c[isl]->f=y;
            c[isl]=y;
            y->pushup();
        }

        //only used for makeroot
        void splay(Node *tg)
        {
            for (Node *y=f;y!=tg;zigzag(),y=f)
                if (Node *z=y->f;z!=tg)
                    (z->c[1]==y^y->c[1]==this?this:y)->zigzag();
            pushup();
        }

        void clear()
        {
            for (Node *x:c)
                if (x)
                    x->clear();
            delete this;
        }
    };

    Node *rt;
    int shift;

    Splay()
    {
        rt=new Node;
        rt->c[1]=new Node;
        rt->c[1]->f=rt;
        rt->siz=2;
    }

    Splay(vector<Info> &a,int l,int r)
    {
        shift=l-1;
        rt=new Node;
        rt->c[1]=new Node;
        rt->c[1]->f=rt;
        if (l<r)
        {
            rt->c[1]->c[0]=build(a,l,r);
            rt->c[1]->c[0]->f=rt->c[1];
        }
        rt->c[1]->pushup();
        rt->pushup();
    }

    Node *build(vector<Info> &a,int l,int r)
    {
        if (l==r) return 0;
        int m=(l+r)>>1;
        Node *x=new Node(a[m]);
        x->c[0]=build(a,l,m);
        x->c[1]=build(a,m+1,r);
        for (Node *y:x->c)
            if (y) y->f=x;
        x->pushup();
        return x;
    }

    void makeroot(Node *u,Node *tg)
    {
        if (!tg) rt=u;
        u->splay();
    }

    void findKth(int k,Node *tg)
    {
        Node *x=rt;
        while (1)
        {
            x->pushdown();
            int res=x->c[0]?x->c[0]->siz:0;
            if (res+1==k)
            {
                x->splay(tg);
                if (!tg) rt=x;
                return;
            }
            if (res>=k) x=x->c[0];
            else x=x->c[1],k-=res+1;
        }
    }

    void split(int l,int r)
    {
        findKth(l,0);
        findKth(r+2,rt);
    }

#ifdef _rev
    void reverse(int l,int r)
    {
        l-=shift;
        r-=shift+1;
        if (l>r) return;
        split(l,r);
        *(rt->c[1]->c[0])+=Tag(1);
    }
#endif

    //insert before pos
    void insert(int pos,Info x)
    {
        pos-=shift;
        split(pos,pos-1);
        rt->c[1]->c[0]=new Node(x);
        rt->c[1]->c[0]->f=rt->c[1];
        rt->c[1]->pushup();
        rt->pushup();
    }

    void insert(int pos,vector<Info> &a,int l,int r)
    {
        pos-=shift;
        split(pos,pos-1);
        rt->c[1]->c[0]=build(a,l,r);
        rt->c[1]->c[0]->f=rt->c[1];
        rt->c[1]->pushup();
        rt->pushup();
    }

    void erase(int pos)
    {
        pos-=shift;
        split(pos,pos);
        delete rt->c[1]->c[0];
        rt->c[1]->c[0]=0;
        rt->c[1]->pushup();
        rt->pushup();
    }

    void erase(int l,int r)
    {
        l-=shift,r-=shift+1;
        if (l>r) return;
        split(l,r);
        rt->c[1]->c[0]->clear();
        rt->c[1]->c[0]=0;
        rt->c[1]->pushup();
        rt->pushup();
    }

    void modify(int pos,Info x)
    {
        pos-=shift;
        findKth(pos+1,0);
        rt->v=x;
        rt->pushup();
    }

    void rangeApply(int l,int r,Tag w)
    {
        l-=shift,r-=shift+1;
        if (l>r) return;
        split(l,r);
        Node *x=rt->c[1]->c[0];
        *x+=w;
        rt->c[1]->pushup();
        rt->pushup();
    }

    Info rangeQuery(int l,int r)
    {
        l-=shift,r-=shift+1;
        split(l,r);
        return rt->c[1]->c[0]->s;
    }

    ~Splay() { rt->clear(); }
#undef _rev
};

struct Tag
{
    bool rev=0;

    Tag() {}
    Tag(bool c):rev(c) {}

    void operator += (const Tag &o)
    {
        rev^=o.rev;
    }
};

struct Info
{
    i64 x=0;

    void operator += (const Tag &o) const
    {

    }

    Info operator + (const Info &o) const
    {
        return {x+o.x};
    }
};
```

图论

拓扑排序

```
vector<int> topo(vector<vector<int>> &adj)
{
    int n=adj.size();
    vector<int> res,in(n);
    queue<int> q;
    for (int u=0;u<n;u++)
        for (int v:adj[u])
            in[v]++;
    for (int u=0;u<n;u++)
        if (!in[u])
            q.push(u);
    while (!q.empty())
    {
        int u=q.front();
        q.pop();
        res.push_back(u);
        for (int v:adj[u])
        {
            in[v]--;
            if (!in[v]) q.push(v);
        }
    }
    return res;
}
```

树的直径

```
int diameter(vector<vector<int>> &adj)
{
    int n=adj.size(),d=0;
    vector<int> dp(n);

    auto dfs=[&](auto &self,int u,int f)->void
    {
        for (int v:adj[u])
        {
            if (v==f) continue;
            self(self,v,u);
            d=max(d,dp[u]+dp[v]+1);//w(u,v)=1
            dp[u]=max(dp[u],dp[v]+1);//w(u,v)=1
        }
    };

    dfs(dfs,0,0);
    return d;
}
```

动态树直径（CF1192B）

指支持动态修改树边的权值，复杂度为 $O(\log n)$。

代码 `d,e->D,E` 那段是题目强制在线的解密。

```
struct Tag
{
    i64 dt=0;
    void apply(Tag t)
    {
        dt+=t.dt;
    }
```

```
};

struct Info
{
    i64 ans=0,mx=0,mn=1e18,lm=0,rm=0;
    void apply(Tag t)
    {
        mx+=t.dt;
        mn+=t.dt;
        lm-=t.dt;
        rm-=t.dt;
    }
};

Info operator + (Info a,Info b)
{
    Info c;
    c.ans=max({a.ans,b.ans,a.rm+b.mx,a.mx+b.lm});
    c.mx=max(a.mx,b.mx);
    c.mn=min(a.mn,b.mn);
    c.lm=max({a.lm,b.lm,b.mx-2*a.mn});
    c.rm=max({a.rm,b.rm,a.mx-2*b.mn});
    return c;
}

void R()
{
    i64 n,q,w;
    cin>>n>>q>>w;
    vector<int> in(n),out(n),ord;
    vector<i64> dep(n,-1);
    vector<array<i64,3>> edges(n-1);
    vector<vector<array<i64,2>>> adj(n);
    for (int i=1;i<n;i++)
    {
        i64 a,b,c;
        cin>>a>>b>>c;
        a--,b--;
        edges[i-1]={a,b,c};
        adj[a].push_back({b,c});
        adj[b].push_back({a,c});
    }

    auto dfs=[&](auto &self,int u)->void
    {
        in[u]=out[u]=ord.size();
        ord.push_back(u);
        for (auto [v,w]:adj[u])
        {
            if (dep[v]!=-1) continue;
            dep[v]=dep[u]+w;
            self(self,v);
            out[u]=ord.size();
            ord.push_back(u);
        }
    };

    dep[0]=0;
    dfs(dfs,0);

    SGT<Info,Tag> sgt(ord.size());
    for (int i=0;i<ord.size();i++)
        sgt.modify(i,{0ll,dep[ord[i]],dep[ord[i]],-dep[ord[i]],-dep[ord[i]]});

    i64 las=0;
    for (int i=0;i<q;i++)
    {
        i64 d,e,D,E;
        cin>>d>>e;
        D=(d+las)%(n-1);
        E=(e+las)%w;
        auto &[x,y,w]=edges[D];
        if (in[x]>in[y]) swap(x,y);
        sgt.rangeApply(in[y],out[y]+1,{E-w});
        w=E;
        cout<<(las=sgt.rangeQuery(0,ord.size()).ans)<<'\n';
    }
    return;
}
```

树的重心

```
vector<int> centroid(vector<vector<int>> &adj,int rt)
{
    int n=adj.size();
    vector<int> siz(n),res(n),w(n),fa(n);

    auto dfs=[&](auto &self,int u,int f)->void
    {
        siz[u]=1,res[u]=u,fa[u]=f;
        for (int v:adj[u])
        {
            if (v==f) continue;
            self(self,v,u);
            siz[u]+=siz[v];
            w[u]=max(w[u],siz[v]);
        }
        for (int v:adj[u])
        {
            if (v==f) continue;
            int p=res[v];
            while (p!=u)
            {
                if (max(w[p],siz[u]-siz[p])<=siz[u]/2)
                {
                    res[u]=p;
                    break;
                }
                else p=fa[p];
            }
        }
    };

    dfs(dfs,rt,rt);
    return res;
}
```

Dijkstra

注意设定合适的 inf。

```
vector<i64> dijk(const vector<vector<pair<int,i64>>> &adj,int s)
{
    int n=adj.size();
    using pa=pair<i64,int>;
    vector<i64> d(n,inf);
    vector<int> ed(n);
    priority_queue<pa,vector<pa>,greater<pa>> q;
    q.push({0,s}); d[s]=0;
    while (!q.empty())
    {
        int u=q.top().second;
        q.pop();
        ed[u]=1;
        for (auto [v,w]:adj[u])
            if (d[u]+w<d[v])
            {
                d[v]=d[u]+w;
                q.push({d[v],v});
            }
        while (!q.empty()&&ed[q.top().second]) q.pop();
    }
    return d;
}
```

SPFA

注意设定合适的 inf。

```
vector<i64> spfa(const vector<vector<pair<int,i64>>> &adj,int s)
{
    int n=adj.size();
    assert(n);
    queue<int> q;
    vector<int> len(n),ed(n);
    vector<i64> d(n,inf);
    q.push(s); d[s]=0;
    while (!q.empty())
    {
        int u=q.front();
        q.pop();
        ed[u]=0;
        for (auto [v,w]:adj[u])
            if (d[u]+w<d[v])
            {
                d[v]=d[u]+w;
                len[v]=len[u]+1;
                if (len[v]>n) return {};
                if (!ed[v]) ed[v]=1,q.push(v);
            }
    }
    return d;
}
```

Johnson

```
vector<vector<i64>> dijk(const vector<vector<pair<int,i64>>> &adj)
{
    vector<vector<i64>> res;
    for (int i=0;i<adj.size();i++)
        res.push_back(dijk(adj,i));
    return res;
}

vector<i64> spfa(const vector<vector<pair<int,i64>>> &adj)
{
    int n=adj.size();
    assert(n);
    queue<int> q;
    vector<int> len(n),ed(n,1);
    vector<i64> d(n);
    for (int i=0;i<n;i++) q.push(i);
    while (!q.empty())
    {
        int u=q.front();
        q.pop();
        ed[u]=0;
        for (auto [v,w]:adj[u])
            if (d[u]+w<d[v])
            {
                d[v]=d[u]+w;
                len[v]=len[u]+1;
                if (len[v]>n) return {};
                if (!ed[v]) ed[v]=1,q.push(v);
            }
    }
    return d;
}

vector<vector<i64>> john(vector<vector<pair<int,i64>>> adj)
{
    int n=adj.size();
    assert(n);
    auto h=spfa(adj);
    if (!h.size()) return {};
    for (int u=0;u<n;u++)
        for (auto &[v,w]:adj[u])
            w+=h[u]-h[v];
    auto res=dijk(adj);
    for (int u=0;u<n;u++)
        for (int v=0;v<n;v++)
            if (res[u][v]!=inf)
                res[u][v]-=h[u]-h[v];
    return res;
}
```

强连通分量

```
struct SCC
{
    int n,cur,cnt;
    vector<vector<int>> adj;
    vector<int> stk,dfn,low,bel;

    SCC() {}
    SCC(int n) { init(n); }

    void init(int n)
    {
        this->n=n;
        adj.assign(n,{});
        stk.clear();
        dfn.assign(n,-1);
        low.resize(n);
        bel.assign(n,-1);
        cur=cnt=0;
    }

    void add(int u,int v) { adj[u].push_back(v); }

    void dfs(int x)
    {
        dfn[x]=low[x]=cur++;
        stk.push_back(x);
        for (auto y:adj[x])
        {
            if (dfn[y]==-1)
            {
                dfs(y);
                low[x]=min(low[x],low[y]);
            }
            else if (bel[y]==-1) low[x]=min(low[x],dfn[y]);
        }
        if (dfn[x]==low[x])
        {
            int y;
            do
            {
                y=stk.back();
                bel[y]=cnt;
                stk.pop_back();
            } while (y!=x);
            cnt++;
        }
    }

    vector<int> work()
    {
        for (int i=0;i<n;i++)
            if (dfn[i]==-1) dfs(i);
        return bel;
    }

    struct Graph
    {
        int n;
        vector<pair<int,int>> edges;
        vector<int> siz,cnte;
    };

    Graph compress()
    {
        Graph G;
        G.n=cnt;
        G.siz.resize(cnt);
        G.cnte.resize(cnt);
        for (int i=0;i<n;i++)
        {
            G.siz[bel[i]]++;
            for (auto j:adj[i])
                if (bel[i]!=bel[j])
                    G.edges.emplace_back(bel[j],bel[i]);
        }
        return G;
    };
};
```

边双连通分量

```
struct EBCC
{
    int n;
    vector<vector<int>> adj;
    vector<int> stk,dfn,low,bel;
    int cur,cnt;

    EBCC() {}
    EBCC(int n) { init(n); }

    void init(int n)
    {
        this->n=n;
        adj.assign(n,{});
        dfn.assign(n,-1);
        low.resize(n);
        bel.assign(n,-1);
        stk.clear();
        cur=cnt=0;
    }

    void add(int u,int v)
    {
        adj[u].push_back(v);
        adj[v].push_back(u);
    }

    void dfs(int x,int p)
    {
        dfn[x]=low[x]=cur++;
        stk.push_back(x);
        for (auto y:adj[x])
        {
            if (y==p) continue;
            if (dfn[y]==-1)
            {
                dfs(y,x);
                low[x]=min(low[x],low[y]);
            }
            else if (bel[y]==-1&&dfn[y]<dfn[x]) low[x]=min(low[x],dfn[y]);
        }
        if (dfn[x]==low[x])
        {
            int y;
            do
            {
                y=stk.back();
                bel[y]=cnt;
                stk.pop_back();
            } while (y!=x);
            cnt++;
        }
    }

    vector<int> work()
    {
        dfs(0,-1);
        return bel;
    }

    struct Graph
    {
        int n;
        vector<pair<int,int>> edges;
        vector<int> siz,cnte;
    };

    Graph compress()
    {
        Graph G;
        G.n=cnt;
        G.siz.resize(cnt);
        G.cnte.resize(cnt);
        for (int i=0;i<n;i++)
        {
            G.siz[bel[i]]++;
            for (auto j:adj[i])
            {
                if (bel[i]<bel[j]) G.edges.emplace_back(bel[i],bel[j]);
                else if (i<j) G.cnte[bel[i]]++;
            }
        }
        return G;
    };
};
```

轻重链剖分

```
struct HLD
{
    int n;
    vector<int> siz,top,dep,pa,in,out,seq;
    vector<vector<int>> adj;
    int cur;

    HLD(){}
    HLD(int n) { init(n); }

    void init(int n)
    {
        this->n=n;
        siz.resize(n);
        top.resize(n);
        dep.resize(n);
        pa.resize(n);
        in.resize(n);
        out.resize(n);
        seq.resize(n);
        cur=0;
        adj.assign(n,{});
    }

    void addEdge(int u,int v)
    {
        adj[u].push_back(v);
        adj[v].push_back(u);
    }

    void work(int rt=0)
    {
        top[rt]=rt;
        dep[rt]=0;
        pa[rt]=-1;
        dfs1(rt);
        dfs2(rt);
    }

    void dfs1(int u)
    {
        if (pa[u]!=-1) adj[u].erase(find(adj[u].begin(),adj[u].end(),pa[u]));
        siz[u]=1;
        for (auto &v:adj[u])
        {
            pa[v]=u;
            dep[v]=dep[u]+1;
            dfs1(v);
            siz[u]+=siz[v];
            if (siz[v]>siz[adj[u][0]])
                swap(v,adj[u][0]);
        }
    }

    void dfs2(int u)
    {
        in[u]=cur++;
        seq[in[u]]=u;
        for (auto v:adj[u])
        {
            top[v]=(v==adj[u][0])?top[u]:v;
            dfs2(v);
        }
        out[u]=cur;
    }

    int lca(int u,int v)
    {
        while (top[u]!=top[v])
        {
            if (dep[top[u]]>dep[top[v]]) u=pa[top[u]];
            else v=pa[top[v]];
        }
        return dep[u]<dep[v]?u:v;
    }

    int dist(int u,int v) { return dep[u]+dep[v]-(dep[lca(u,v)]<<1); }

    int jump(int u,int k)
    {
        if (dep[u]<k) return -1;
        int d=dep[u]-k;
        while (dep[top[u]]>d) u=pa[top[u]];
        return seq[in[u]-dep[u]+d];
    }

    bool isAncester(int u,int v) { return in[u]<=in[v]&&in[v]<out[u]; }

    int rootedParent(int u,int v)//u->root,v->point
    {
        if (u==v) return u;
        if (!isAncester(v,u)) return pa[v];
        auto it=upper_bound(adj[v].begin(),adj[v].end(),u,[&](int x,int y){ return in[x]<in[y]; })-1;
        return *it;
    }

    int rootedSize(int u,int v)//same as rootedParent
    {
```

```cpp
        if (u==v) return n;
        if (!isAncester(v,u)) return siz[v];
        return n-siz[rootedParent(u,v)];
    }

    int rootedLca(int a,int b,int c) { return lca(a,b)^lca(b,c)^lca(c,a); }
};
```

虚树

```cpp
struct VirtualTree
{
    int n,rt;
    HLD hld;
    vector<int> a;
    vector<bool> is;
    vector<vector<int>> son;

    VirtualTree(){}
    VirtualTree(int n) { init(n); }

    void init(int n)
    {
        this->n=n;
        hld.init(n);
        is.assign(n,0);
        son.assign(n,{});
    }

    void addEdge(int u,int v)
    {
        hld.addEdge(u,v);
    }

    void work(int rt=0)
    {
        this->rt=rt;
        hld.work(rt);
    }

    void solve(vector<int> &in)
    {
        auto cmp=[&](int x,int y)->bool
        {
            return hld.in[x]<hld.in[y];
        };

        for (int x:a)
        {
            is[x]=0;
            son[x].clear();
        }
        a=in;
        for (int x:a) is[x]=1;
        a.push_back(rt);
        sort(a.begin(),a.end(),cmp);

        int k=a.size();
        for (int i=1;i<k;i++)
            a.push_back(hld.lca(a[i-1],a[i]));
        sort(a.begin(),a.end(),cmp);
        a.erase(unique(a.begin(),a.end()),a.end());
        for (int i=1;i<a.size();i++)
            son[hld.lca(a[i-1],a[i])].push_back(a[i]);
    };

    bool isKey(int u)
    {
        return is[u];
    }

    vector<int>& operator [] (int u)
    {
        return son[u];
    }
};
```

欧拉路径

```cpp
vector<int> euler(vector<vector<int>> adj)
{
    int n=adj.size(),x=0;
    vector<int> in(n),out(n);
    for (int u=0;u<n;u++)
        for (int v:adj[u])
            out[u]++,in[v]++;
    for (int i=0;i<n;i++)
        if (in[i]!=out[i])
        {
            if (abs(in[i]-out[i])>1) return {};
            x++;
        }
    if (x>2) return {};
    for (int i=0;i<n;i++)
        if (out[i]>in[i])
        {
            x=i;
            break;
        }
    for (int i=0;i<n;i++)
        sort(adj[i].begin(),adj[i].end(),greater<int>());

    vector<int> res;
    auto dfs=[&](auto &self,int u)->void
    {
        while (!adj[u].empty())
        {
            int v=adj[u].back();
            adj[u].pop_back();
            self(self,v);
            res.push_back(v);
        }
    };

    dfs(dfs,x);
    res.push_back(x);
    reverse(res.begin(),res.end());
    return res;
}
```

2-SAT

```cpp
struct TwoSat
{
    int n;
    vector<vector<int>> e;
    vector<bool> ans;

    TwoSat(int n):n(n),e(n<<1),ans(n){}

    void addClause(int u,bool f,int v,bool g)
    {
        e[u*2+!f].push_back(v*2+g);
        e[v*2+!g].push_back(u*2+f);
    }

    bool satisfiable()
    {
        vector<int> id(n*2,-1),dfn(n*2,-1),low(n*2,-1),stk;
        int now=0,cnt=0;
        function<void(int)> tarjan=[&](int u)
        {
            stk.push_back(u);
            dfn[u]=low[u]=now++;
            for (auto v:e[u])
            {
                if (dfn[v]==-1)
                {
                    tarjan(v);
                    low[u]=min(low[u],low[v]);
                }
                else if (id[v]==-1)
                    low[u]=min(low[u],dfn[v]);
            }
            if (dfn[u]==low[u])
            {
                int v;
                do
                {
                    v=stk.back();
                    stk.pop_back();
                    id[v]=cnt;
                } while (v!=u);
                cnt++;
            }
        };
        for (int i=0;i<n*2;i++)
            if (dfn[i]==-1)
                tarjan(i);
        for (int i=0;i<n;i++)
        {
            if (id[i*2]==id[i*2+1]) return 0;
            ans[i]=id[i*2]>id[i*2+1];
        }
        return 1;
    }
    vector<bool> answer() { return ans; }
};
```

最大流

```
template <class T>
struct MaxFlow
{
    struct _Edge
    {
        int to;
        T cap;
        _Edge(int to,T cap):to(to),cap(cap){}
    };

    int n;
    vector<_Edge> e;
    vector<vector<int>> g;
    vector<int> cur,h;

    MaxFlow(){}
    MaxFlow(int n) { init(n); }

    void init(int n)
    {
        this->n=n;
        e.clear();
        g.assign(n,{});
        cur.resize(n);
        h.resize(n);
    }

    bool bfs(int s,int t)
    {
        h.assign(n,-1);
        queue<int> que;
        h[s]=0;
        que.push(s);
        while (!que.empty())
        {
            const int u=que.front();
            que.pop();
            for (int i:g[u])
            {
                auto [v,c]=e[i];
                if (c>0&&h[v]==-1)
                {
                    h[v]=h[u]+1;
                    if (v==t) return 1;
                    que.push(v);
                }
            }
        }
        return 0;
    }

    T dfs(int u,int t,T f)
    {
        if (u==t) return f;
        auto r=f;
        for (int &i=cur[u];i<int(g[u].size());i++)
        {
            const int j=g[u][i];
            auto [v,c]=e[j];
            if (c>0&&h[v]==h[u]+1)
            {
                auto a=dfs(v,t,min(r,c));
                e[j].cap-=a;
                e[j^1].cap+=a;
                r-=a;
                if (r==0) return f;
            }
        }
        return f-r;
    }

    void addEdge(int u,int v,T c)
    {
        g[u].push_back(e.size());
        e.emplace_back(v,c);
        g[v].push_back(e.size());
        e.emplace_back(u,0);
    }

    T flow(int s,int t)
    {
        T ans=0;
        while (bfs(s,t))
        {
            cur.assign(n,0);
            ans+=dfs(s,t,numeric_limits<T>::max());
        }
        return ans;
    }

    vector<bool> minCut()
    {
        vector<bool> c(n);
        for (int i=0;i<n;i++) c[i]=(h[i]!=-1);
        return c;
    }

    struct Edge
    {
        int from;
        int to;
        T cap;
        T flow;
    };

    vector<Edge> edges()
    {
        vector<Edge> a;
        for (int i=0;i<e.size();i+=2)
        {
            Edge x;
            x.from=e[i+1].to;
            x.to=e[i].to;
            x.cap=e[i].cap+e[i+1].cap;
            x.flow=e[i+1].cap;
            a.push_back(x);
        }
        return a;
    }
};
```

最小费用最大流

```
template <class T>
struct MinCostFlow
{
    struct _Edge
    {
        int to;
        T cap;
        T cost;
        _Edge(int to,T cap,T cost):to(to),cap(cap),cost(cost){}

    };

    int n;
    vector<_Edge> e;
    vector<vector<int>> g;
    vector<T> h,dis;
    vector<int> pre;

    bool john(int s,int t)
    {
        dis.assign(n,numeric_limits<T>::max());
        pre.assign(n,-1);
        priority_queue<pair<T,int>,vector<pair<T,int>>,greater<pair<T,int>>> q;
        dis[s]=0;
        q.emplace(0,s);
        while (!q.empty())
        {
            T d=q.top().first;
            int u=q.top().second;
            q.pop();
            if (dis[u]!=d) continue;
            for (int i:g[u])
            {
                int v=e[i].to;
                T cap=e[i].cap;
                T cost=e[i].cost;
                if (cap>0&&dis[v]>d+h[u]-h[v]+cost)
                {
                    dis[v]=d+h[u]-h[v]+cost;
                    pre[v]=i;
                    q.emplace(dis[v],v);
                }
            }
        }
        return dis[t]!=numeric_limits<T>::max();
    }

    MinCostFlow(){}
    MinCostFlow(int n) { init(n); }

    void init(int n_)
    {
        n=n_;
        e.clear();
        g.assign(n,{});
    }

    void addEdge(int u,int v,T cap,T cost)
    {
        g[u].push_back(e.size());
        e.emplace_back(v,cap,cost);
        g[v].push_back(e.size());
        e.emplace_back(u,0,-cost);
    }

    pair<T,T> flow(int s,int t)
    {
        T flow=0;
        T cost=0;
        h.assign(n,0);
        while (john(s,t))
        {
            for (int i=0;i<n;i++) h[i]+=dis[i];
            T aug=numeric_limits<int>::max();
            for (int i=t;i!=s;i=e[pre[i]^1].to)
                aug=min(aug,e[pre[i]].cap);
            for (int i=t;i!=s;i=e[pre[i]^1].to)
            {
                e[pre[i]].cap-=aug;
                e[pre[i]^1].cap+=aug;
            }
            flow+=aug;
            cost+=aug*h[t];
        }
        return make_pair(flow,cost);
    }

    struct Edge
    {
        int from;
        int to;
        T cap;
        T cost;
        T flow;
    };

    vector<Edge> edges()
    {
        vector<Edge> a;
        for (int i=0;i<e.size();i+=2)
        {
            Edge x;
            x.from=e[i+1].to;
            x.to=e[i].to;
            x.cap=e[i].cap+e[i+1].cap;
            x.cost=e[i].cost;
            x.flow=e[i+1].cap;
            a.push_back(x);
        }
        return a;
    }
};
```

二分图最大权匹配（KM）

时间复杂度为 $O(n^3)$。

```
//注意将负权边加上 inf，inf 不要设得过大
//xy 是左部点对应右部点
//yx 是右部点对应左部点
template <class T>
```

```
struct MaxAssignment
{
    vector<T> lx,ly,s,cst;
    vector<int> xy,yx,p,sx;
    vector<bool> visx,visy;

    T solve(int nx,int ny,vector<vector<T>> a)
    {
        assert(0<=nx&&nx<=ny);
        assert(int(a.size())==nx);
        for (int i=0;i<nx;i++)
        {
            assert(int(a[i].size())==ny);
            for (auto x:a[i])
                assert(x>=0);
        }
        auto upd=[&](int x)->void
        {
            for (int y=0;y<ny;y++)
            {
                if (lx[x]+ly[y]-a[x][y]<s[y])
                {
                    s[y]=lx[x]+ly[y]-a[x][y];
                    sx[y]=x;
                }
            }
            return;
        };
        cst.resize(nx+1);
        cst[0]=0;
        lx.assign(nx,numeric_limits<T>::max());
        ly.assign(ny,0);
        xy.assign(nx,-1);
        yx.assign(ny,-1);
        sx.resize(ny);
        for (int cur=0;cur<nx;cur++)
        {
            queue<int> q;
            visx.assign(nx,0);
            visy.assign(ny,0);
            s.assign(ny,numeric_limits<T>::max());
            p.assign(nx,-1);
            for (int x=0;x<nx;x++)
            {
                if (xy[x]==-1)
                {
                    q.push(x);
                    visx[x]=1;
                    upd(x);
                }
            }
            int ex,ey;
            bool fl=0;
            while (!fl)
            {
                while (!q.empty()&&!fl)
                {
                    auto x=q.front();
                    q.pop();
                    for (int y=0;y<ny;y++)
                    {
                        if (a[x][y]==lx[x]+ly[y]&&!visy[y])
                        {
                            if (yx[y]==-1)
                            {
                                ex=x;
                                ey=y;
                                fl=1;
                                break;
                            }
                            q.push(yx[y]);
                            p[yx[y]]=x;
                            visy[y]=visx[yx[y]]=1;
                            upd(yx[y]);
                        }
                    }
                }
                if (fl) break;
                T delta=numeric_limits<T>::max();
                for (int y=0;y<ny;y++)
                    if (!visy[y])
                        delta=min(delta,s[y]);
                for (int x=0;x<nx;x++)
                    if (visx[x])
                        lx[x]-=delta;
                for (int y=0;y<ny;y++)
                {
                    if (visy[y])
                        ly[y]+=delta;
                    else
                        s[y]-=delta;
                }
                for (int y=0;y<ny;y++)
                {
                    if (!visy[y]&&s[y]==0)
                    {
                        if (yx[y]==-1)
                        {
                            ex=sx[y];
                            ey=y;
                            fl=1;
                            break;
                        }
                        q.push(yx[y]);
                        p[yx[y]]=sx[y];
                        visy[y]=visx[yx[y]]=1;
                        upd(yx[y]);
                    }
                }
            }
            cst[cur+1]=cst[cur];
            for (int x=ex,y=ey,ty;x!=-1;x=p[x],y=ty)
            {
                cst[cur+1]+=a[x][y];
                if (xy[x]!=-1)
                    cst[cur+1]-=a[x][xy[x]];
                ty=xy[x];
                xy[x]=y;
                yx[y]=x;
            }
        }
        return cst[nx];
    }

    vector<int> assignment() { return xy; }

    pair<vector<T>,vector<T>> labels()
    { return make_pair(lx,ly); }

    vector<T> weights() { return cst; }
};
```

三元环计数

时间复杂度为 $\mathcal{O}(m\sqrt{m})$。

```
i64 triple(vector<pair<int,int>> &edges)
{
    int n=0;
    for (auto [u,v]:edges) n=max({n,u,v});
    n++;
    vector<int> d(n),id(n),rk(n),cnt(n);
    vector<vector<int>> adj(n);
    for (auto [u,v]:edges) d[u]++,d[v]++;
    iota(id.begin(),id.end(),0);
    sort(id.begin(),id.end(),[&](int x,int y)
    {
        return d[x]<d[y];
    });
    for (int i=0;i<n;i++) rk[id[i]]=i;
    for (auto [u,v]:edges)
    {
        if (rk[u]>rk[v]) swap(u,v);
        adj[u].push_back(v);
    }
    i64 res=0;
    for (int i=0;i<n;i++)
    {
        for (int u:adj[i]) cnt[u]=1;
        for (int u:adj[i])
            for (int v:adj[u])
                res+=cnt[v];
        for (int u:adj[i]) cnt[u]=0;
    }
    return res;
};
```

树哈希

有根树返回各子树hash值，无根树返回一个至多长为2的vector。

```
vector<int> tree_hash(vector<vector<int>> &adj,int rt)
{
    int n=adj.size();
    static map<vector<int>,i64> mp;
    static int id=0;
    vector<int> h(n);

    auto dfs=[&](auto &self,int u,int f)->void
    {
        vector<int> c;
        for (int v:adj[u])
            if (v!=f)
            {
                self(self,v,u);
                c.push_back(h[v]);
            }
        sort(c.begin(),c.end());
        if (!mp.count(c)) mp[c]=id++;
        h[u]=mp[c];
    };

    dfs(dfs,rt,rt);
    return h;
}

vector<int> tree_hash(vector<vector<int>> &adj)
{
    int n=adj.size();
    if (n==0) return {};
    vector<int> siz(n),mx(n);

    auto dfs=[&](auto &self,int u)->void
    {
        siz[u]=1;
        for (int v:adj[u])
            if (!siz[v])
            {
                self(self,v);
                siz[u]+=siz[v];
                mx[u]=max(mx[u],siz[v]);
            }
        mx[u]=max(mx[u],n-siz[u]);
    };

    dfs(dfs,0);
    int m=*min_element(mx.begin(),mx.end());
    vector<int> rt;
    for (int i=0;i<n;i++)
        if (mx[i]==m)
            rt.push_back(i);
    for (int &u:rt) u=tree_hash(adj,u)[u];
    sort(rt.begin(),rt.end());
    return rt;
}
```

矩阵树定理

记度矩阵为 D，邻接矩阵为 A。

对无向图情况：$L(G)=D(G)-A(G)$。

对有向图外向树情况：$L(G)=D^{in}(G)-A(G)$。

对有向图内向树情况：$L(G)=D^{out}(G)-A(G)$。

图 G 以 r 为根的生成树个数等于 $L(G)$ 舍去第 r 行第 r 列的 $n-1$ 阶主子式。

代码中 `t=0` 是无向图情况，`t=1` 是有向图根为 1 的外向树情况。

```
void R()
{
    int n,m,t;
    cin>>n>>m>>t;
    vector<vector<Z>> L(n-1,vector<Z>(n-1)),D(n,vector<Z>(n)),A(n,vector<Z>(n));;
    for (int i=1;i<=m;i++)
    {
        int u,v,w;
        cin>>u>>v>>w;
        if (u==v) continue;
        u--,v--;
        D[v][v]+=w;
        A[u][v]+=w;
        if (t==0)
        {
            D[u][u]+=w;
            A[v][u]+=w;
        }
    }
    for (int i=1;i<n;i++)
        for (int j=1;j<n;j++)
            L[i-1][j-1]=D[i][j]-A[i][j];
    cout<<det(L);
    return;
}
```

计算几何

EPS

```
const double eps=1e-8;
int sgn(double x)
{
    if (fabs(x)<eps) return 0;
    if (x>0) return 1;
    return -1;
}
```

Point

```
template <class T>
struct Point
```

```cpp
{
    T x,y;
    Point(T x_=0,T y_=0):x(x_),y(y_) {}

    Point &operator += (Point p) &
    {
        x+=p.x;
        y+=p.y;
        return *this;
    }

    Point &operator -= (Point p) &
    {
        x-=p.x;
        y-=p.y;
        return *this;
    }

    Point &operator *= (T v) &
    {
        x*=v;
        y*=v;
        return *this;
    }

    Point operator - () const { return Point(-x,-y); }

    friend Point operator + (Point a,Point b) { return a+=b; }
    friend Point operator - (Point a,Point b) { return a-=b; }
    friend Point operator * (Point a,T b) { return a*=b; }
    friend Point operator * (T a,Point b) { return b*=a; }

    friend bool operator == (Point a,Point b) { return a.x==b.x&&a.y==b.y; }

    friend istream &operator >> (istream &is,Point &p) { return is>>p.x>>p.y; }

    friend ostream &operator << (ostream &os,Point p) { return os<<'('<<p.x<<','<<p.y<<')'; }
};

template <class T>
int sgn(const Point<T> &a) { return a.y>0||(a.y==0&&a.x>0)?1:-1; }

template <class T>
T dot(Point<T> a,Point<T> b) { return a.x*b.x+a.y*b.y; }

template <class T>
T cross(Point<T> a,Point<T> b) { return a.x*b.y-a.y*b.x; }

template <class T>
T square(Point<T> p) { return dot(p,p); }

template <class T>
double length(Point<T> p) { return sqrt(double(square(p))); }

long double length(Point<long double> p) { return sqrt(square(p)); }
```

Line

```cpp
template <class T>
struct Line
{
    Point<T> a,b;
    Line(Point<T> a_=Point<T>(),Point<T> b_=Point<T>()):a(a_),b(b_) {}
};
```

距离

```cpp
template <class T>
double dis_PP(Point<T> a,Point<T> b) { return length(a-b); }

```

```
template <class T>
double dis_PL(Point<T> a,Line<T> l) { return fabs(cross(a-l.a,a-l.b))/dis_PP(l.a,l.b); }

template <class T>
double dis_PS(Point<T> a,Line<T> l)
{
    if (dot(a-l.a,l.b-l.a)<0) return dis_PP(a,l.a);
    if (dot(a-l.b,l.a-l.b)<0) return dis_PP(a,l.b);
    return dis_PL(a,l);
}
```

点绕中心旋转

```
template <class T>
Point<T> rotate(Point<T> a,double alpha)
{ return Point<T>(a.x*cos(alpha)-a.y*sin(alpha),a.x*sin(alpha)+a.y*cos(alpha)); }
```

关于线的对称点

```
template <class T>
Point<T> lineRoot(Point<T> a,Line<T> l)
{
    Point<T> v=l.b-l.a;
    return l.a+v*(dot(a-l.a,v)/dot(v,v));
}

template <class T>
Point<T> symmetry_PL(Point<T> a,Line<T> l) { return a+(lineRoot(a,l)-a)*2; }
```

位置关系判断

```
template <class T>
bool pointOnSegment(Point<T> a,Line<T> l)
{ return (sgn(cross(a-l.a,a-l.b))==0)&&(sgn(dot(a-l.a,a-l.b))<=0); }

template <class T>
bool lineCrossLine(Line<T> a,Line<T> b)
{
    double f1=cross(b.a-a.a,a.b-a.a),f2=cross(b.b-a.a,a.b-a.a);
    double g1=cross(a.a-b.a,b.b-b.a),g2=cross(a.b-b.a,b.b-b.a);
    return ((f1<0)^(f2<0))&&((g1<0)^(g2<0));
}

template <class T>
bool pointOnLineLeft(Point<T> a,Line<T> l) { return cross(l.b-l.a,a-l.a)>0; }

//适用任意多边形,O(n)
template <class T>
bool pointInPolygon(Point<T> a,const vector<Point<T>> &p)
{
    int n=p.size();
    for (int i=0;i<n;i++)
        if (pointOnSegment(a,Line<T>(p[i],p[(i+1)%n])))
            return 1;
    bool t=0;
    for (int i=0;i<n;i++)
    {
        Point<T> u=p[i],v=p[(i+1)%n];
        if (u.x<a.x&&v.x>=a.x&&pointOnLineLeft(a,Line<T>(v,u))) t^=1;
        if (u.x>=a.x&&v.x<a.x&&pointOnLineLeft(a,Line<T>(u,v))) t^=1;
    }
    return t;
}

//适用凸多边形,O(log n)
template <class T>
bool pointInPolygon_(Point<T> a,const vector<Point<T>> &p)
{
    int n=p.size();
    if (cross(a-p[0],p[1]-p[0])<0||cross(a-p[0],p[n-1]-p[0])>0) return 0;
    if (pointOnSegment(a,Line<T>(p[0],p[1]))||pointOnSegment(a,Line<T>(p[n-1],p[0]))) return 1;
    int l=1,r=n-1;
    while (l+1<r)
    {
        int mid=(l+r)>>1;
        if (cross(a-p[1],p[mid]-p[1])<0) l=mid;
        else r=mid;
    }
    if (cross(a-p[l],p[r]-p[l])>0) return 0;
    if (pointOnSegment(a,Line<T>(p[l],p[r]))) return 1;
    return 1;
}
```

线段交点

```
//小心平行
template <class T>
Point<T> lineIntersection(Line<T> a,Line<T> b)
{
    Point<T> u=a.a-b.a,v=a.b-a.a,w=b.b-b.a;
    double t=cross(u,w)/cross(w,v);
    return a.a+t*v;
}
```

过定点做圆的切线

```
template <class T>
vector<Line<T>> tan_PC(Point<T> a,Point<T> c,T r)
{
    Point<T> v=c-a;
    vector<Line<T>> res;
    int dis=dis_PP(a,c);
    if (sgn(dis-r)==0) res.push_back(rotate(v,acos(-1)/2));
    else if (dis>r)
    {
        double alpha=asin(r/dis);
        res.push_back(rotate(v,alpha));
        res.push_back(rotate(v,-alpha));
    }
    return res;
}
```

两圆交点

```
template <class T>
vector<Point<T>> circleIntersection(Point<T> c1,T r1,Point<T> c2,T r2)
{
    auto get=[&](Point<T> c,T r,double alpha)->Point<T>
    { return Point<T>(c.x+cos(alpha)*r,c.y+sin(alpha)*r); };

    auto angle=[&](Point<T> a)->double { return atan2(a.x,a.y); };

    vector<Point<T>> res;
    double d=dis_PP(c1,c2);
    if (sgn(d)==0) return res;
    if (sgn(r1+r2-d)<0) return res;
    if (sgn(fabs(r1-r2)-d)>0) return res;
    double alpha=angle(c2-c1);
    double beta=acos((r1*r1-r2*r2+d*d)/(r1*d*2));
    Point<T> p1=get(c1,r1,alpha-beta),p2=get(c1,r1,alpha+beta);
    res.push_back(p1);
    if (p1!=p2) res.push_back(p2);
    return res;
}
```

多边形面积

```cpp
template <class T>
double polygonArea(const vector<Point<T>> &p)
{
    int n=p.size();
    double res=0;
    for (int i=1;i<n-1;i++) res+=cross(p[i]-p[0],p[i+1]-p[0]);
    return fabs(res/2);
}
```

自适应辛普森法

```cpp
//注意边界函数值不能小于 eps
double f(double x) { return pow(x,0.5); }
double calc(double l,double r)
{
    double mid=(l+r)/2.0;
    return (r-l)*(f(l)+f(r)+f(mid)*4.0)/6.0;
}
double simpson(double l,double r,double lst)
{
    double mid=(l+r)/2.0;
    double fl=calc(l,mid),fr=calc(mid,r);
    if (sgn(fl+fr-lst)==0) return fl+fr;
    else return simpson(l,mid,fl)+simpson(mid,r,fr);
}
```

静态凸包

```cpp
template <class T>
vector<Point<T>> getHull(vector<Point<T>> p)
{
    vector<Point<T>> h,l;
    sort(p.begin(),p.end(),[&](auto a,auto b)
    {
        if (a.x!=b.x) return a.x<b.x;
        else return a.y<b.y;
    });
    p.erase(unique(p.begin(),p.end()),p.end());
    if (p.size()<=1) return p;
    for (auto a:p)
    {
        while (h.size()>1&&sgn(cross(a-h.back(),a-h[h.size()-2]))<=0) h.pop_back();
        while (l.size()>1&&sgn(cross(a-l.back(),a-l[l.size()-2]))>=0) l.pop_back();
        l.push_back(a);
        h.push_back(a);
    }
    l.pop_back();
    reverse(h.begin(),h.end());
    h.pop_back();
    l.insert(l.end(),h.begin(),h.end());
    return l;
}
```

旋转卡壳求直径

```cpp
template <class T>
double getDiameter(vector<Point<T>> p)
{
    double res=0;
    if (p.size()==2) return dis_PP(p[0],p[1]);
    int n=p.size();
    p.push_back(p.front());
    int j=2;
    for (int i=0;i<n;i++)
    {
        while (sgn(cross(p[i+1]-p[i],p[j]-p[i])-cross(p[i+1]-p[i],p[j+1]-p[i]))<0)
            j=(j+1)%n;
        res=max({res,dis_PP(p[i],p[j]),dis_PP(p[i+1],p[j])});
    }
    return res;
}
```

半平面交

```
template <class T>
vector<Point<T>> hp(vector<Line<T>> lines)
{
    sort(lines.begin(),lines.end(),[&](auto l1,auto l2)
    {
        auto d1=l1.b-l1.a;
        auto d2=l2.b-l2.a;

        if (sgn(d1)!=sgn(d2)) return sgn(d1)==1;
        return cross(d1,d2)>0;
    });

    deque<Line<T>> ls;
    deque<Point<T>> ps;
    for (auto l:lines)
    {
        if (ls.empty())
        {
            ls.push_back(l);
            continue;
        }
        while (!ps.empty()&&!pointOnLineLeft(ps.back(),l))
        {
            ps.pop_back();
            ls.pop_back();
        }
        while (!ps.empty()&&!pointOnLineLeft(ps[0],l))
        {
            ps.pop_front();
            ls.pop_front();
        }
        if (cross(l.b-l.a,ls.back().b-ls.back().a)==0)
        {
            if (dot(l.b-l.a,ls.back().b-ls.back().a)>0)
            {
                if (!pointOnLineLeft(ls.back().a,l))
                {
                    assert(ls.size()==1);
                    ls[0]=l;
                }
                continue;
            }
            return {};
        }
        ps.push_back(lineIntersection(ls.back(),l));
        ls.push_back(l);
    }
    while (!ps.empty()&&!pointOnLineLeft(ps.back(),ls[0]))
    {
        ps.pop_back();
        ls.pop_back();
    }
    if (ls.size()<=2) return {};
    ps.push_back(lineIntersection(ls[0],ls.back()));
    return vector(ps.begin(),ps.end());
}
```

最小圆覆盖

期望时间复杂度为 $\mathcal{O}(n)$。

```
using Real=long double;

//only for 3*3
```

```cpp
Real det(vector<vector<Real>> a)
{
    Real res=0;
    for (int i=0;i<3;i++)
    {
        Real tmp=1;
        for (int j=0;j<3;j++)
            tmp*=a[j][(i+j)%3];
        res+=tmp;
    }
    for (int i=0;i<3;i++)
    {
        Real tmp=1;
        for (int j=0;j<3;j++)
            tmp*=a[j][(i+j*2)%3];
        res-=tmp;
    }
    return res;
}

mt19937_64 rnd(chrono::steady_clock::now().time_since_epoch().count());

tuple<Point<Real>,Real> Coverage(vector<Point<Real>> p)
{
    int n=p.size();
    shuffle(p.begin(),p.end(),rnd);
    Point<Real> C=p[0];
    Real r=0;
    for (int i=0;i<n;i++)
        if (dis_PP(C,p[i])>r)
        {
            C=p[i],r=0;
            for (int j=0;j<i;j++)
                if (dis_PP(C,p[j])>r)
                {
                    C=(p[i]+p[j])*0.5;
                    r=dis_PP(p[i],p[j])*0.5;
                    for (int k=0;k<j;k++)
                        if (dis_PP(C,p[k])>r)
                        {
                            array<Real,3> x,y;
                            x[0]=p[i].x,y[0]=p[i].y;
                            x[1]=p[j].x,y[1]=p[j].y;
                            x[2]=p[k].x,y[2]=p[k].y;
                            vector<vector<Real>> a(3,vector<Real>(3)),b(a),c(a);
                            for (int t=0;t<3;t++)
                            {
                                a[t][0]=b[t][0]=x[t]*x[t]+y[t]*y[t];
                                c[t][0]=b[t][1]=x[t];
                                a[t][1]=c[t][1]=y[t];
                                a[t][2]=b[t][2]=c[t][2]=1;
                            }
                            Real px=det(a)/det(c)/2.0,py=-det(b)/det(c)/2.0;
                            C={px,py};
                            r=dis_PP(C,p[i]);
                        }
                }
        }
    return {C,r};
}
```