代码中默认使用了以下宏和库：

```
#include <stdio.h>
#include <iostream>
#include <algorithm>
#include <cmath>
#include <string.h>
#include <assert.h>
using namespace std;

#define re register
#define ms(x, vl) memset(x, vl, sizeof(x))
#define mc(x, y, sz) memcpy(x, y, (sz)*sizeof(int))
#define db double
#define ll long long
#define _for(i, a, b) for(int i = (a); i < (b); ++i)
#define _rfor(i, a, b) for(int i = (a); i <= (b); ++i)
#define _dfor(i, b, a) for(int i = (b); i >= (a); --i)
#define maxn 1000000
```

目录

数据结构

链表

指针版

```
struct Node{ Node *pre, *suf; /*...*/};

Node *add_node(Node *p, Node nn){
    Node *q = new Node(nn);
    q->suf = p->suf; q->pre = p; p->suf = q;
    if (q->suf != 0) q->suf->pre = q;
    return q;
}

void del_node(Node *p){
    assert(p != 0);
    if (p->pre != 0) p->pre->suf = p->suf;
    if (p->suf != 0) p->suf->pre = p->pre;
    delete p;
}
```

内存池版

```
#define maxnn 1000005
int pcnt = 0,
    pool[maxnn];
struct Node{ int pre, suf; /*...*/ } node[maxnn];
#define pre(x) node[x].pre
#define suf(x) node[x].suf
```

```
void init_pool(){
    _for(i, 1, maxnn) pool[++pcnt] = maxnn-i;
}

int new_node(int nn){
    int x = pool[pcnt--];
    node[x] = node[nn];
    return x;
}

int add_node(int p, int nn){
    int q = new_node(nn);
    suf(q) = suf(p); pre(q) = p; suf(p) = q;
    if (suf(q)) pre(suf(q)) = q;
    return q;
}

void del_node(int p){
    if (pre(p)) suf(pre(p)) = suf(p);
    if (suf(p)) pre(suf(p)) = pre(p);
    pool[++pcnt] = p;
}
```

树状数组

```
int n, fwk[maxn];

void add_fwk(int x, int vl){ for(; x <= n; x += x&-x) fwk[x] += vl; }

int qry_fwk(int x){ int vl = 0; for(; x; x -= x&-x) vl += fwk[x]; return vl; }
```

线段树

```
int rg;
struct Seg{
    /*...*/
    int tag;
}seg[maxn<<2];
#define lx (x<<1)
#define rx (x<<1|1)
#define tag(x) seg[x].tag

void ud(re int x){
    /*...*/
}

void build(int x, int tl, int tr){
    /*...*/
    if (tl == tr){
        /*...*/
        return;
    }
    int mi = (tl+tr)>>1;
    build(lx, tl, mi);
    build(rx, mi+1, tr);
    ud(x);
}
```

```
void spread(int x){
    /*...*/
}

void pd(re int x){
    assert(x); //注意不要对 0 节点 pushdown
    spread(lx);
    spread(rx);
    tag(x) = /*...*/;
}

//区间操作
void func(int x, /*...*/, int l, int r, int tl, int tr){
    if (tl == l && tr == r){ spread(x); return; }
    if (tag[x] != /*...*/) pd(x);
    int mi = (tl+tr)>>1;
    if (r <= mi) func(lx, /*...*/, l, r, tl, mi);
    else if (l > mi) func(rx, /*...*/, l, r, mi+1, tr);
    else func(lx, /*...*/, l, mi, tl, mi), func(rx, /*...*/, mi+1, r, mi+1, tr);
    ud(x);
}

//单点操作
int stk[maxn];
void func(int p, /*...*/){
    re int x = 1, tl = 1, tr = rg, mi, top = 0;
    while(tl < tr){
        stk[top++] = x;
        int mi = (tl+tr)>>1;
        if (p <= mi) x = lx, tr = mi;
        else x = rx, tl = mi+1;
    }
    /*...*/
    while(top) ud(stk[--top]);
}
```

线段树二分

```
int qry(int x, int l, int r, int tl, int tr){
    if (tr < l || tl > r) return -1;
    if (l <= tl && r >= tr){
        /*...*/;
        return /*...*/;
    }
    int mi = (tl+tr)>>1, tmp = qry(lx, l, r, tl, mi);
    return (tmp > 0) ? tmp : qry(rx, l, r, mi+1, tr);
}
```

动态开点线段树

模板与线段树模板基本一致，只有些许不同

```
int pcnt,
    pool[maxnn];
struct Seg{
    int lc, rc;
    /*...*/
}seg[maxnn];
#define lc(x) seg[x].lc
```

```
#define rc(x) seg[x].rc

void init(){
    lc(0) = rc(0) = 0;
    _for(i, 1, maxnn) pool[++pcnt] = maxnn-i;
}

int new_node(re int nn){
    re int x = pool[pcnt--];
    seg[x] = seg[nn];
    return x;
}

void del(re int x){
    pool[++pcnt] = x;
}

void ud(re int x){
    /*...*/
}

void build(int x, int tl, int tr){
    x = new_node(0);
    if (tl == tr){
        /*...*/
        return;
    }
    int mi = (tl+tr)>>1;
    build(lx, tl, mi);
    build(rx, mi+1, tr);
    ud(x);
}

void spread(int x){
    /*...*/
}

void pd(re int x){
    assert(x); //注意不要对 0 节点 pushdown
    if (lc(x)) spread(lc(x));
    if (rc(x)) spread(rc(x));
    tag(x) = /*...*/;
}

//区间操作
void func(int &x, /*...*/, int l, int r, int tl, int tr){
    if (!x) x = new_node(0);
    if (tl == l && tr == r){ spread(x); return; }
    if (tag[x] != /*...*/) pd(x);
    int mi = (tl+tr)>>1;
    if (r <= mi) func(lx, /*...*/, l, r, tl, mi);
    else if (l > mi) func(rx, /*...*/, l, r, mi+1, tr);
    else func(lx, /*...*/, l, mi, tl, mi), func(rx, /*...*/, mi+1, r, mi+1, tr);
    ud(x);
}

//单点操作
void func(int &x, /*...*/, int p, int tl, int tr){
    if (!x) x = new_node(0);
    if (tl == tr){ /*...*/; return; }
    int mi = (tl+tr)>>1
    if (p <= mi) func(lc(x), /*...*/, p, tl, mi);
    else func(rc(x), /*...*/, p, mi+1, tr);
    ud(x);
}
```

线段树分裂、合并

线段树合并一定是动态开点线段树，可根据需要选择是否回收废弃节点。

```
//按值域分裂
void vl_split(int x, int p, int &ltr, int &rtr, int tl, int tr){
    if (!x){ ltr = rtr = 0; return; }
    assert(tl < tr);
    int mi = (tl+tr)>>1;
    ltr = x; rtr = new_node(x);
    if (p == mi) rc(ltr) = lc(rtr) = 0;
    else if (p < mi){
        rc(ltr) = 0;
        vl_split(lc(x), p, lc(ltr), lc(rtr), tl, mi);
    }
    else{
        lc(rtr) = 0;
        vl_split(rc(x), p, rc(ltr), rc(ltr), mi+1, tr);
    }
    ud(ltr); ud(rtr);
}

//按排名分裂
void rk_split(int x, int rk, int &ltr, int &rtr, int tl, int tr){
    assert(rk >= 1 && rk < sz(x));
    int mi = (tl+tr)>>1;
    ltr = x; rtr = new_node(x);
    if (tl == tr){ sz(ltr) = rk; sz(rtr) -= rk; retrun; }
    if (rk == sz(lc(x))) rc(ltr) = lc(rtr) = 0;
    else if (rk < sz(lc(x))){
        rc(ltr) = 0;
        rk_split(lc(x), rk, lc(ltr), lc(rtr), tl, mi);
    }
    else{
        lc(rtr) = 0;
        rk_split(rc(x), rk-sz(lc(x)), rc(ltr), rc(rtr), mi+1, tr);
    }
    ud(ltr); ud(rtr);
}

//合并并返回新根
int merge(int x, int y, /*...*/, int tl, int tr){
    if (!x || !y){
        if (x || y) spread(x?x:y, /*...*/);
        return x+y;
    }
    if (tl == tr){
        /*...*/
        del(y);
        return x;
    }
    int mi = (tl + tr)>>1;
    lc(x) = merge(lc(x), lc(y), /*...*/, tl, mi);
    rc(x) = merge(rc(x), rc(y), /*...*/, mi+1, tr);
    ud(x); del(y);
    return x;
}
```

主席树

```
int ncnt = 0, rg;
struct Seg{
    int lc, rc;
    /*...*/
}seg[maxnn];
#define lc(x) seg[x].lc
#define rc(x) seg[x].rc

int new_node(int nn){
    re int x = ++ncnt;
    seg[x] = seg[nn];
    return x;
}

void ud(int x){
    /*...*/
}

//单点修改
//x 是历史版本，y 是新增版本
void func(int x, int &y, /*...*/, int p, int tl, int tr){
    y = new_node(x);
    if (tl == tr){ /*...*/ return; }
    int mi = (tl+tr)>>1;
    if (p <= mi) func(lc(x), lc(y), /*...*/, tl, mi);
    else func(rc(x), rc(y), /*...*/, mi+1, tr);
    ud(y);
}

void spread(int x){
    /*...*/
}

//区间修改，标记永久化技巧
//x 是历史版本，y 是新增版本
void func(int x, int &y, /*...*/, int l, int r, int tl, int tr){
    y = new_node(x);
    if (l == tl && r == tr){ spread(y); return; }
    int mi = (tl+tr)>>1;
    if (r <= mi) func(lc(x), lc(y), /*...*/, l, r, tl, mi);
    else if (l > mi) func(rc(x), rc(y), /*...*/, l, r, mi+1, tr);
    else func(lc(x), lc(y), /*...*/, l, mi, tl, mi), func(rc(x), rc(y), /*...*/, mi+1, r, mi+1, tr);
    ud(y);
}
```

线段树优化建图

```
int ecnt, ncnt, rg,
    head[maxnn], to[maxm], nex[maxm],
    vl[maxnn], tmp_arr[2][maxnn];
#define lx (x<<1)
#define rx (x<<1|1)

int new_node(){
    int x = ++ncnt;
    head[x] = 0;
    return x;
}

void init_seg(int rg_){ rg = rg_; ecnt = 0; ncnt = 0; new_node(); }

void ae(int u, int v){ to[++ecnt] = v; nex[ecnt] = head[u]; head[u] = ecnt; }

void build(int x, int tl, int tr){
    vl[x] = new_node(); new_node(); assert(ncnt == vl[x]^1);
    if (tl == tr){
        /*...*/
        return;
    }
    int mi = (tl+tr)>>1;
    build(lx, tl, mi);
    build(rx, mi+1, tr);
    ae(vl[x], vl[lx]); ae(vl[x], vl[rx]);
    ae(vl[lx]^1, vl[x]^1); ae(vl[rx]^1, vl[x]^1);

}

int scur;
void find(int x, int *arr, int l, int r, int tl, int tr){
    if (l <= tl && tr <= r){ arr[++scur] = x; return; }
    int mi = (tl+tr)>>1;
    if (l <= mi) find(lx, arr, l, r, tl, mi);
    if (r > mi) find(rx, arr, l, r, mi+1, tr);
}

void add_edge(int l, int r, int L, int R){
    int *t = tmp_arr[0], *T = tmp_arr[1], d, D;
    scur = 0; find(1, t, l, r, 1, rg); d = scur;
    scur = 0; find(1, T, L, R, 1, rg); D = scur;
    _rfor(i, 1, d) _rfor(j, 1, D) ae(vl[t[i]]^1, vl[T[j]]);
}
```

树链剖分

```
int ecnt = 1, dfn_clk = 0;
struct Edge{ int to, nex, we; } edge[2*maxn];
struct Node{ int head, fa, sz, se, dep, ddp, top, btm, pre, pst; } node[maxn];
#define to(x) edge[x].to
#define nex(x) edge[x].nex
#define we(x) edge[x].we
#define head(x) node[x].head
#define fa(x) node[x].fa
#define sz(x) node[x].sz
#define se(x) node[x].se
```

```
#define dep(x) node[x].dep
#define ddp(x) node[x].ddp
#define top(x) node[x].top
#define btm(x) node[x].btm
#define pre(x) node[x].pre
#define pst(x) node[x].pst
#define tpu top(u)
#define tpv top(v)
#define ptu pre(top(u))
#define ptv pre(top(v))

int get_son(int u){
    sz(u) = 1; se(u) = 0;
    _fev(p, u){
        int v = to(p);
        if (v == fa(u)) continue;
        fa(v) = u;
        dep(v) = dep(u) + 1;
        ddp(v) = ddp(u) + we(p);
        dfs(v);
        sz(u) += sz(v);
        if (!se(u) || sz(v) > sz(to(se(u)))) se(u) = p;
    }
}

int subdivide(int u, int tp){
    pre(u) = ++dfn_clk; top(u) = tp;
    if (se(u)) subdivide(to(se(u)), tp), btm(u) = btm(to(se(u)));
    else btm(u) = u;
    _fev(p, u) if (p != se(u) && to(p) != fa(u)) subdivide(to(p), to(p));
    pst(u) = dfn_clk;
}

int get_lca(int u, int v, int &dis){
    dis = ddp(u) + ddp(v);
    while(tpu != tpv){
        if (dep(tpu) < dep(tpv)) u^=v^=u^=v;
        u = fa(tpu);
    }
    if (dep(u) > dep(v)) u^=v^=u^=v;
    dis -= 2*ddp(u);
    return u;
}
```

ChthollyTree

```
int n;
struct Node{
    int l, r;
    mutable int v;
    Node(int l = 0, int r = 0, int v = 0):l(l), r(r), v(v){}
    Node(const Node &t):l(t.l), r(t.r), v(t.v){}
    inline bool operator<(const Node &t)const{ return l < t.l }
};
set<Node> ct;

void init(int n_){
    n = n_;
    ct.clear();
    ct.insert(Node(1, n, 0));
}

auto find(int p){
    if (p > n) return ct.end();
    return --ct.upper_bound(Node(p, 0, 0));
}

auto split(int p){
    auto it = find(p);
    if (it == ct.end() || it->l == p) return it;
    Node nn(*it);
    ct.erase(it);
    ct.insert(Node(nn.l, p-1, nn.v));
    nn.l = p;
    return ct.insert(nn).first;
}

//注意先split(r+1)再split(l)
//反了则 itr 可能会失效以至于 RE
auto assign(int l, int r, int v){
    auto = itr = split(r+1), itl = split(l);
    ct.erase(itl, itr);
    return ct.insert((Node){l, r, v}).first;
}

void performance(int l, int r, /*...*/){
    auto itr = split(r+1), itl = split(l);
    for(; itl != itr; ++itl){
        /*...*/
    }
}
```

Dsu on Tree

```
#define maxn ...
int ecnt = 1,
    head[maxn],
    to[maxn*2],
    nex[maxn*2],
    sz[maxn],
    son[maxn];
bool vis[maxn];

void add_edge(int u, int v);

void get_son(int u, int fa){
    sz[u] = 1; son[u] = 0;
    _fev(p, u){
        int v = to[p];
        if (v == fa || vis[v]) continue;
        get_son(v, u);
        sz[u] += sz[v];
        if (!son[u] || sz[v] > sz[son[u]]) son[u] = v;
    }
}

void adn(int u){
    /*...*/
}

void add_tree(int u, int fa){
    adn(u);
    /*...*/
    _fev(p, u) if (to[p] != fa) add_tree(to[p], u);
}

void clr(int u, int fa){
    /*...*/
    _fev(p, u) if (to[p] != fa) clr(to[p], u);
}

//在调用 dsu(1, 0) 前要调用 get_son(1, 0)
void dsu(int u, int fa, bool fg = false){
    _fev(p, u) if (to[p] != fa && to[p] != son[u]) dsu(to[p], u, false);
    if (son[u]) dsu(son[u], u, true);
    adn(u);
    _fev(p, u){
        int v = to[p];
        if (v == fa || v == son[u]) continue;
        add_tree(v, u);
    }
    /*...*/
    if (!fg) clr(u, fa);
}
```

点分治

```
#define maxn ...
int ecnt = 1,
    head[maxn],
    to[maxn*2],
    nex[maxn*2],
    sz[maxn],
    son[maxn];
bool vis[maxn];

void add_edge(int u, int v);

void get_son(int u, int fa){
    sz[u] = 1; son[u] = 0;
    _fev(p, u){
        int v = to[p];
        if (v == fa || vis[v]) continue;
        get_son(v, u);
        sz[u] += sz[v];
        if (!son[u] || sz[v] > sz[son[u]]) son[u] = v;
    }
}

int get_centroid(int u){
    if (!son[u]) return u;
    int S = sz[u], v;
    while(sz[v = son[u]]*2 > S) u = v;
    return u;
}
```

```
/*
一些函数
*/

void divide(int u){
    assert(!vis[u]);
    get_son(u, 0); u = get_centroid(u); vis[u] = true;

    _fev(p, u){
        int v = to[p];
        if (vis[v]) continue;
        /*...*/
    }
    /*...*/

    _fev(p, u) if (!vis[to[p]]) divide(to[p]);
}
```

边分治\边分树

```
#define maxn /*...*/
#define lbmaxn /*...*/
struct DTree{
    int ncnt, ecnt,
        head[maxn*2],
        to[maxn*4],
        nex[maxn*4],
        we[maxn*4],
        in[maxn*2],
        dep[maxn*2],
        se[maxn*2],
        sz[maxn*2],
        hei[maxn*2],
        bny[maxn*2],
        val[maxn*2][lbmaxn];
    bool ban[maxn*4];
    void init(int n){ this->n = n; ecnt = 1; _rfor(i, 1, 2*n) head[i] = in[i] = 0; }
    void ae(int u, int v, int w){
        to[++ecnt] = v; nex[ecnt] = head[u]; head[u] = ecnt;
        ++in[u]; we[ecnt] = w; ban[ecnt] = false;
        to[++ecnt] = u; nex[ecnt] = head[v]; head[v] = ecnt;
        ++in[v]; we[ecnt] = w;ban[ecnt] = false;
    }
    void spilt(int u){ //三度化的分裂操作
        int p = head[u], q = nex[p], r = nex[q];
        head[u] = r; nex[q] = 0; in[u] = 1;
        head[++ncnt] = p; in[ncnt] = 2;
        to[p^1] = to[q^1] = ncnt;
        ae(u, ncnt, 0);
    }
    void add_edge(int u, int v, int w){
        if (in[u] == 3) spilt(u);
        if (in[v] == 3) spilt(v);
        ae(u, v, w);
    }
    void get_son(int u, int fa, int h = -1){
        if (~h){ val[u][h] = dep[u]; /*...*/ }
        sz[u] = 1; se[u] = 0;
        _fev(p, u){
            int v = to[p];
            if (ban[p] || v == fa) continue;
            dep[v] = dep[u] + we[p];
            get_son(v, u);
            sz[u] += sz[v];
            if (!se[u] || sz[v] > sz[to[se[u]]]) se[u] = p;
        }
    }
    int get_centroid(int u){
        if (!se[u]) return u;
        int S = sz[u], v;
        while(sz[v = to[se[u]]]*2 > S) u = v;
        return u;
    }
    //在调用 divide(1) 前先要调用 get_son(1, 0, -1)
    void divide(int u, int h, int vl){
        u = get_centroid(u);
        if (!se[u]){ hei[u] = h; bny[u] = vl; return; }
        ban[se[u]] = ban[se[u]^1] = true;
        int v = to[se[u]];
        dep[u] = 0; dep[v] = we[p];
        get_dep(u, 0, h); get_dep(v, 0, h);
        divide(u, h+1, vl);
        divide(v, h+1, vl|(1<<h));
    }
}
```

KD-Tree

以下均是二维 KD-Tree 的实现。

静态 KD-Tree 树

```
struct Point{ int x, y; /*...*/ } pnt[maxn];
struct Range{ int xl, xr, yl, yr; };
struct KDNode{ Range rg; /*...*/ } kdt[maxn<<2];
#define rg(x) kdt[x].rg
#define xl(x) kdt[x].rg.xl
#define xr(x) kdt[x].rg.xr
#define yl(x) kdt[x].rg.yl
#define yr(x) kdt[x].rg.yr
#define lx (x<<1)
#define rx (x<<1|1)

inline bool cmpx(const Point &p1, const Point &p2){ return p1.x < p2.x; }
inline bool cmpy(const Point &p1, const Point &p2){ return p1.y < p2.y; }

inline void ud(int x){
    rg(x) = {
        min(xl(lx), xl(rx)), max(xr(lx), xr(rx)),
        min(yl(lx), yl(rx)), max(yr(lx), yr(rx))
    };
    /*...*/
}

//轮换选轴法
void build(int x, int tl, int tr, bool kd = false){
    if (tl == tr){ rg(x) = {pnt[tl].x, pnt[tl].x, pnt[tl].y, pnt[tl].y}; return; }
    int mi = (tl+tr)>>1;
    nth_element(pnt+tl, pnt+mi, pnt+tr+1, kd?cmpy:cmpx);
    build(lx, tl, mi, !kd);
    build(rx, mi+1, tr, !kd);
    ud(x);
}

//方差选轴法
void build(int x, int tl, int tr){
    if (tl == tr){ rg(x) = {pnt[tl].x, pnt[tl].x, pnt[tl].y, pnt[tl].y}; return; }
    int mi = (tl+tr)>>1;
    db ax = 0.0, ay = 0.0, vx = 0.0, vy = 0.0;
    _rfor(i, tl, tr) ax += pnt[i].x, ay += pnt[i].y; ax /= tr-tl+1; ay /= tr-tl+1;
    _rfor(i, tl, tr) vx += (pnt[i].x-ax)*(pnt[i].x-ax), vy += (pnt[i].y-ay)*(pnt[i].y-
ay);
    nth_element(pnt+tl, pnt+mi, pnt+tr+1, vy>vx?cmpy:cmpx);
    build(lx, tl, mi);
    build(rx, mi+1, tr);
    ud(x);
}

//对矩阵的查询
Range q; int qas; /*...*/
void qry(int x, int tl, int tr){
    if (xr(x)<q.xl || xl(x)>q.xr || yr(x)<q.yl || yl(x)>q.yr) return;
    if (xl(x)>=q.xl && xr(x)<=q.xr && yl(x)>=q.yl && yr(x)<=q.yr){ /*...*/ return; }
    int mi = (tl+tr)>>1;
    qry(lx, tl, mi);
    qry(rx, mi+1, tr);
}
```

动态 KD-Tree

```
#include <vector>

int pcnt, pool[maxnn];
struct Point{ int x, y; /*...*/ } pnt[maxn];
struct Range{ int xl, xr, yl, yr; };
struct KDNode{ int lc, rc; Range rg; /*...*/ } kdtnode[maxnn] = {{0, 0, {0, 0, 0}}};

void init_pool(int n){ pcnt = 0; _rfor(i, 1, n) pool[++pcnt] = n-i+1; }
int new_node(){ int x = pool[pcnt--]; kdtnode[x] = kdtnode[0]; return x; }
int del(int x){ pool[++pcnt] = x; }

inline bool cmpx(const Point &p1, const Point &p2){ return p1.x < p2.x; }
inline bool cmpy(const Point &p1, const Point &p2){ return p1.y < p2.y; }
inline bool inner(const Point &p, const Range &rg){
    return p.x>=rg.xl && p.x<=rg.xr && p.y>=rg.yl && p.y<=rg.yr;
}

struct KDT{
    #define rg(x) kdtnode[x].rg
    #define xl(x) kdtnode[x].rg.xl
    #define xr(x) kdtnode[x].rg.xr
    #define yl(x) kdtnode[x].rg.yl
    #define yr(x) kdtnode[x].rg.yr
    #define lc(x) kdtnode[x].lc
    #define rc(x) kdtnode[x].rc
    int rt;
    vector<Point> *pnt_;
    #define pnt (*pnt_)
    inline void ud(int x){
        rg(x) = {
            min(xl(lx), xl(rx)), max(xr(lx), xr(rx)),
            min(yl(lx), yl(rx)), max(yr(lx), yr(rx))
        };
        /*...*/
    }
    void build(int &x, int tl, int tr){
        if (!x) x = new_node();
        if (tl == tr){ rg(x) = {pnt[tl].x, pnt[tl].x, pnt[tl].y, pnt[tl].y}; return; }
        int mi = (tl+tr)>>1;
        db ax = 0.0, ay = 0.0, vx = 0.0, vy = 0.0;
        _rfor(i, tl, tr) ax += pnt[i].x, ay += pnt[i].y; ax /= tr-tl+1; ay /= tr-tl+1;
        _rfor(i, tl, tr)
            vx += (pnt[i].x-ax)*(pnt[i].x-ax), vy += (pnt[i].y-ay)*(pnt[i].y-ay);
        nth_element(pnt.begin()+tl, pnt.begin()+mi, pnt.begin()+tr+1, vy>vx?cmpy:cmpx);
        build(lc(x), tl, mi);
        build(rc(x), mi+1, tr);
        ud(x);
    }
    Range q; int qas; /*...*/
    void qry(int x, int tl, int tr){
        if (xr(x)<q.xl || xl(x)>q.xr || yr(x)<q.yl || yl(x)>q.yr) return;
        if (xl(x)>=q.xl && xr(x)<=q.xr && yl(x)>=q.yl && yr(x)<=q.yr){ /*...*/ return;
}
        int mi = (tl+tr)>>1;
        qry(lc(x), tl, mi);
        qry(rc(x), mi+1, tr);
    }
    void clr(int x){
        if (!x) return;
        clr(lc(x));
        clr(rc(x));
        del(x);
    }
    void clear(){
        clr(rt);
        rt = 0;
    }
    #undef rg
    #undef xl
    #undef xr
    #undef yl
    #undef yr
    #undef lc
    #undef rc
};

struct KDTree{
    KDT T0, T1; int p0, p1, p2;
    vector<Point> pnt;
    init(){ pnt.clear(); pnt.push_back(Point()); p0 = p1 = p2 = 0; T0.pnt = T1.pnt =
&pnt; }
    void ins(const Point &point){
        pnt.push_back(point); ++p2;
        if ((p2-p1)*(p2-p1) > p2*1.5){
            T0.clear();
            T0.build(T0.rt, p0+1, p1=p2);
        }
        else if (p1-p0 > pow(p2, 0.75)*1.2){
            T0.clear(); T1.clear();
            T1.build(T1.rt, 1, p0=p2-1);
            T0.build(T0.rt, p2, p1=p2);
        }
    }
    //对矩阵的查询
    int qry(const Range &rg){
        int ans;
        T0.q = T1.q = rg; T0.qas = T1.qas = 0;
        if (p1 > p0) T0.qry(1, p0+1, p1);
        if (p0 > 1) T1.qry(1, 1, p0);
        _rfor(i, p1+1, p2){
            if (inner(pnt[i], rg)){
                /*...*/
            }
        }
        return ans;
    }
};
```

虚树

```
int get_lca(int u, int v);
bool cmp(int a, int b){ return dfn[a] < dfn[b]; }

struct VTree{
    int ecnt, ncnt,
        head[maxn], to[maxn], nex[maxn], fa[maxn], stk[maxn];
    VTree(){ ecnt = ncnt = 0; }
    void add_edge(int u, int v){
        to[++ecnt] = v; nex[ecnt] = head[u]; head[u] = ecnt;
    }

    int build(vector<int> vec){
        int top = ecnt = ncnt = 0;
        sort(vec.begin(), vec.end(), cmp);
        head[stk[++top] = 1] = 0;
        for(auto u : vec){
            if (u == 1) continue;
            int lca = get_lca(u, stk[top]);
            if (lca != stk[top]){
                while(dfn[lca] < dfn[stk[top-1]])
                    add_edge(stk[top-1], stk[top]), --top;
                if (lca != stk[top-1])
                    //三条语句的顺序不能错
                    head[lca] = 0, add_edge(lca, stk[top]), stk[top] = lca;
                else add_edge(lca, stk[top--]);
            }
            head[stk[++top] = u] = 0;
        }
        while(top > 1) add_edge(stk[top-1], stk[top]), --top;
    }
};
```

字符串

字符串 hash

```
int p = 131, epw[maxn], hsh[maxn];

void prepare_hsh(int n, char *str){
    epw[0] = 1;
    _rfor(i, 1, n){
        epw[i] = (ll)epw[i-1]*p%mod;
        hsh[i] = ((ll)hsh[i-1]*p+str[i])%mod;
    }
}

int get_hsh(int i, int j){
    assert(j >= i);
    return (hsh[j]+mod-(ll)hsh[i]*epw[j-i]%mod)%mod;
}
```

AC 自动机

小字符集版

```
#define maxc 26

char str[maxn];
int ncnt = 1,
    nxt[maxn][maxc],
    fail[maxn],
    que[maxn];

void insert(char *str){
    int u = 1, c;
    while(c = *str++){
        if (!nxt[u][c-='a']) nxt[u][c] = ++ncnt;
        u = nxt[u][c];
    }
    /*...*/
}

void build(){
    int u, v, fro = 0, bac = 0;
    _for(i, 0, maxc)
        if (v = nxt[1][i]) fail[que[bac++] = v] = 1;
        else nxt[1][i] = 1;
    while(bac > fro){
        u = que[fro++];
        _for(i, 0, maxc)
            if (v = nxt[u][i]) fail[que[bac++] = v] = nxt[fail[u]][i];
            else nxt[u][i] = nxt[fail[u]][i];
    }
}
```

大字符集版（主席树优化）

```
#define maxc 10000

int ncnt = 1, rcnt = 0, rg = maxc,
    str[maxn],
    nxt[maxn],
    fail[maxn],
    que[maxn];
struct Seg{int lc, rc, tvl; } seg[maxnn];
#define lc(x) seg[x].lc
#define rc(x) seg[x].rc
#define tvl(x) seg[x].tvl

int new_node(int nn){
    re int x = ++rcnt;
    seg[x] = seg[nn];
    return x;
}

void build_seg(int &x, int tl, int tr){
    x = new_node(0);
    if (tl == tr){ tvl(x) = 1; return; }
    int mi = (tl+tr)>>1
    build_seg(lc(x), tl, mi);
    build_seg(rc(x), mi+1, tr);
}

void chgn(int x, int &y_, int vl, int p){
    int y = y_ = new_node(x), tl = 1, tr = rg, mi;
    while(tl != tr){
        mi = (tl+tr)>>1;
        if (p <= mi) x = lc(x), y = lc(y) = new_node(x), tr = mi;
        else x = rc(x), y = rc(y) = new_node(x), tl = mi+1;
    }
    tvl(y) = vl;
}

int qry(int x, int p){
    int tl = 1, tr = rg, mi;
    while(tl != tr){
        mi = (tl+tr)>>1;
        if (p <= mi) x = lc(x), tr = mi;
        else x = rc(x), tl = mi+1;
    }
    return tvl(x);
}

void insert(re int *str){
    re int u = 1, c;
    while(c = *str++){
        if (!mp[u].count(c)) u = mp[u][c] = ++ncnt;
        else u = mp[u][c];
    }
    /*...*/
}

void build(){
    re int u, v, fro = 0, bac = 0;
    build_seg(nxt[1], 1, rg);
    for(auto it : mp[u]){
        fail[que[bac++] = v = it.second] = 1;
        chgn(nxt[1], nxt[1], v, it.first);
    }
    while(bac > fro){
        u = que[fro++];
        nxt[u] = nxt[fail[u]];
        for(auto it : mp[u]){
            fail[que[bac++] = v = it.second] = qry(nxt[u], it.first);
            chgn(nxt[u], nxt[u], v, it.first);
        }
    }
}
```

后缀数组

```
#define maxn 500005
#define lbmaxn 24
int n,
    lb[maxn],
    buc[maxn],
    sa[maxn],
    rk[maxn*2],
    temp_arr[maxn*2],
    ht[lbmaxn][maxn];
void get_sa(char *str, int n, int m = 128){
    ms(rk, 0); ms(temp_arr, 0);
    int *x = rk, *y = temp_arr, p;
    _rfor(i, 1, m) buc[i] = 0;
    _rfor(i, 1, n) ++buc[x[i] = str[i]+1];
    _rfor(i, 2, m) buc[i] += buc[i-1];
    for(int i = n; i >= 1; --i) sa[buc[x[i]]--] = i;
    for(int j = 1; j < n; j <<= 1){
        p = 0;
        _rfor(i, n-j+1, n) y[++p] = i;
        _rfor(i, 1, n) if (sa[i] > j) y[++p] = sa[i]-j;

        _rfor(i, 1, m) buc[i] = 0;
        _rfor(i, 1, n) ++buc[x[i]];
        _rfor(i, 2, m) buc[i] += buc[i-1];
        for(int i = n; i >= 1; --i) sa[buc[x[y[i]]]--] = y[i];

        swap(x, y);
        p = x[sa[1]] = 1;
        _rfor(i, 2, n)
            if (y[sa[i]] == y[sa[i-1]] && y[sa[i]+j] == y[sa[i-1]+j]) x[sa[i]] = p;
            else x[sa[i]] = ++p;
        if (p == n) break;
        else m = p;
    }
    _rfor(i, 1, n) rk[sa[i]] = i;
}

void get_ht(char *str, int n){
    ms(ht, 0);
    int j = 0;
    ht[0][1] = 0;
    _rfor(i, 1, n){
        if (rk[i] == 1) continue;
        if (j) --j;
        char *s1 = str+i-1, *s2 = str+sa[rk[i]-1]-1;
        while(s1[j+1] == s2[j+1]) ++j; //为了防止 j 越界要求字符数组最后一位的下一位与之前的字符均不相同
        ht[0][rk[i]] = j;
    }
}

void prepare(int n){
    lb[0] = -1;
    _rfor(i, 1, n) lb[i] = lb[i>>1]+1;
    _rfor(i, 1, lb[n]){
        _rfor(j, 1, n-(1<<i)+1){
            ht[i][j] = min(ht[i-1][j], ht[i-1][j+(1<<i-1)]);
        }
    }
}

int qry_st(int l, int r){
    int lblen = lb[r-l+1];
    return min(ht[lblen][l], ht[lblen][r-(1<<lblen)+1]);
}

int get_lcp(int l, int r){
    if (l == r) return n-l+1;
    if (rk[l] > rk[r]) l^=r^=l^=r;
    return qry_st(rk[l]+1, rk[r]);
}
```

后缀自动机

```
#define maxn /*...*/
#define maxc 26
int ncnt = 0, sam_lst = 0,
    nxt[maxn*2][maxc], len[maxn*2], fa[maxn*2];

void ins(char *str){
    ncnt = 1; len[1] = fa[1] = 0;
    int c, p, q, np, nq;
    while(c = *str++){
        c -= 'a';
        p = lst; np = ++ncnt; lst = np;
        len[np] = len[p] + 1;
        for(; p && !nxt[p][c]; p = fa[p]) nxt[p][c] = np;
        if (!p) fa[np] = 1;
        else{
            q = nxt[p][c];
            if (len[q] == len[p]+1) fa[np] = q;
            else{
                nq = ++ncnt;
                len[nq] = len[p]+1; fa[nq] = fa[q];
                _for(i, 0, maxc) nxt[nq][i] = nxt[q][i];
                for(; p && nxt[p][c] == q; p = fa[p]) nxt[p][c] = nq;
                fa[q] = fa[np] = nq;
            }
        }
    }
}
```

广义后缀自动机

```
#define maxn /*...*/
#define maxc 26
int ncnt = 1,
    nxt[maxn*2][maxc], fa[maxn*2], len[maxn*2], pos[maxn];
void ins(char *str){
    int u = 1, c;
    while(c = *str++){
        if (!nxt[u][c-='a']) nxt[u][c] = ++ncnt;
        u = nxt[u][c];
    }
}
void build(){
    static int que[maxnn][3]; int fro = 0, bac = 0;
    pos[1] = 1; que[bac][0] = 1; que[bac][1] = 0; que[bac][2] = 0; ++bac;
    while(fro < bac){
        int u = que[fro][0], c = que[fro][1], p = que[fro][2]; ++fro;
        _for(i, 0, maxc) if (nxt[u][i]) que[bac][0] = nxt[u][i], que[bac][1] = i,
que[bac][2] = u, ++bac;
        _for(i, 0, maxc) nxt[u][i] = 0;
        if (u == 1) continue;

        p = pos[p];
        int np = u; pos[u] = np; len[np] = len[p]+1;
        for(; p && !nxt[p][c]; p = fa[p]) nxt[p][c] = np;
        if (!p) fa[np] = 1;
        else{
            int q = nxt[p][c];
            if (len[q] == len[p]+1) fa[np] = q;
            else{
                int nq = ++ncnt; len[nq] = len[p]+1; fa[nq] = fa[q];
                _for(i, 0, maxc) nxt[nq][i] = nxt[q][i];
                for(; p && nxt[p][c] == q; p = fa[p]) nxt[p][c] = nq;
                fa[q] = fa[np] = nq;
            }
        }
    }
}
```

Manacher

```
char mstr[maxn], str[2*maxn];
int rad[2*maxn];

void manacher(int n){
    n = 2*n-1;
    str[0] = '('; str[n+1] = ')';
    _rfor(i, 1, n) str[i] = (i&1) ? mstr[(i+1)>>1] : '#';
    int mid = rad[1] = 1, r = 2;
    _rfor(i, 2, n){
        if (i < r) rad[i] = min(rad[2*mid-i], r-i);
        else rad[i] = 1;
        while(str[i+rad[i]] == str[i-rad[i]]) ++rad[i];
        if (i+rad[i] > r) mid = i, r = i+rad[i];
    }
    _rfor(i, 1, n) rad[i] = rad[i]/2;
}
```

回文自动机

```
int ncnt, curs, lst,
    str[maxn],
    nxt[maxn][maxc], fail[maxn], len[maxn];

void init(){
    ncnt = 2; curs = 0; lst = 2;
    fail[1] = 0; len[1] = -1;
    fail[2] = 1; len[2] = 0;
}

int up(int p){
    while(str[curs-1-len[p]] != str[curs]) p = fail[p];
    return p;
}

void ins(int c){
    str[++curs] = c;
    int p = up(lst), &q = nxt[p][c];
    if (!q){
        q = ++ncnt;
        len[q] = len[p]+2;
        fail[q] = p == 1 ? 2 : nxt[up(fail[p])][c];
    }
    lst = q;
}
```

图论

SPFA

```
int dis[maxn];

bool SPFA(int n, int s){
    static int cnt[maxn], que[maxn], inque[maxn];
    int fro = 0, bac = 0;
    _rfor(i, 1, n) dis[i] = inf, cnt[i] = 0, inque[i] = -1; dis[s] = 0;
    inque[que[bac] = s] = bac; ++bac;
    while(fro != bac){
        int u = que[fro]; fro = (fro+1)%maxn;
        inque[u] = -1;
        _fev(p, u){
            int v = to[p];
            if (dis[u]+we[p] < dis[v]){
                dis[v] = dis[u]+we[p];
                if (inque[v] == -1){
                    inque[que[bac] = v] = bac, bac = (bac+1)%maxn;
                    if (++cnt[v] > n) return false;
                }
            }
            if (dis[v] < dis[que[fro]]){
                inque[que[fro]] = inque[v];
                que[inque[v]] = que[fro];
                que[fro] = v;
                inque[v] = fro;
            }
        }
    }
    return true;
}
```

点双连通分量

```
void tarjan(int u, int fa){
    pre[u] = low[u] = ++dfn_clk;
    stk[top++] = u;
    int v, child = 0;
    _fev(p, u){
        v = to[p];
        if (v == fa) continue;
        if (!pre[v]){
            ++child;
            tarjan(v, u);
            if (low[v] < low[u]) low[u] = low[v];
            if (low[v] >= pre[u]){
                iscut[u] = true;
                bcc[++bcnt].clear();
                re int x;
                do{
                    x = stk[--top];
                    bcc[bcnt].push_back(x);
                }while(x != v);
                bcc[bcnt].push_back(u);
            }
        }
        else if (pre[v] < low[u]) low[u] = pre[v];
    }
    if (!fa){
        if (child == 1) iscut[u] = false;
        else if (child == 0){
            bcc[++bcnt].clear();
            bcc[bcnt].push_back(u);
        }
        top = 0;
    }
}
```

边双连通分量

```
void get_bcc(re int v){
    bcc[++bcnt].clear();
    re int x;
    do{
        x = stk[--top];
        bcc[bcnt].push_back(x);
    }while(x != v);
}

void tarjan(int u, int fa){
    pre[u] = low[u] = ++dfn_clk;
    stk[top++] = u;
    int v;
    _fev(p, u){
        if (p/2 == fa/2) continue;
        v = to[p];
        if (!pre[v]){
            tarjan(v, p);
            if (low[v] < low[u]) low[u] = low[v];
            if (low[v] > pre[u]) get_bcc(v);
        }
        else if (pre[v] < low[u]) low[u] = pre[v];
    }
    if (fa == 0) get_bcc(u);
}
```

强连通分量（缩点）

```
void tarjan(int u){
    pre[u] = low[u] = ++dfn_clk;
    stk[top++] = u;
    int v;
    _fev(p, u){
        v = to[p];
        if (!pre[v]) tarjan(v);
        if (!sd[v] && low[v] < low[u]) low[u] = low[v];
    }
    if (low[u] == pre[u]){
        ++ccnt;
        do{
            v = stk[--top];
            sd[v] = ccnt;
        }while(v != u);
    }
}
```

2-SAT

```
struct TwoSAT{
    bool mark[maxn*2];
    int ecnt = 1, top,
        head[maxn*2],
        to[maxm],
        nex[maxm],
        stk[maxn*2];
    void init(){ ms(head, 0); ms(mark, false); ecnt = 1; }
    void add_edge(int u, int v){
        to[++ecnt] = head[u]; nex[ecnt] = head[u]; head[u] = ecnt;
    }
    void add(int x, int vx, int y, int vy){
        add_edge(x<<1|(!vx), y<<1|vy);
        add_edge(y<<1|(!vy), x<<1|vx);
    }
    bool dfs(int x){
        if (mark[x]) return true;
        if (mark[x^1]) return false;
        mark[x] = true;
        stk[top++] = x;
        _fev(p, x) if (!dfs(to[p])) return false;
        return true;
    }
    bool solve(int n){
        _rfor(i, 1, n){
            if (mark[i<<1] || mark[i<<1|1]) continue;
            top = 0;
            if (dfs(i<<1)) continue;
            while(top) mark[stk[--top]] = false;
            if (!dfs(i<<1|1)) return false;
        }
        return true;
    }
}
```

生成树

Krukal

```
#define maxn 100005
#define maxm 200005
struct Edge{
    int u, v, w;
    inline bool operator<(const Edge &t)const{ return w < t.w; }
} ed[maxm];
int ecnt = 1,
    head[maxn],
    to[maxn*2],
    nex[maxn*2],
    we[maxn*2],
    uf[maxn],
    stk[maxn];

void add_edge(int u, int v, int w){
    to[++ecnt] = v; nex[ecnt] = head[u]; head[u] = ecnt; we[ecnt] = w;
    to[++ecnt] = u; nex[ecnt] = head[v]; head[v] = ecnt; we[ecnt] = w;
}

int find(int u){
    int top = 0;
    while(uf[u] > 0) stk[top++] = u, u = uf[u];
    while(top > 0) uf[stk[--top]] = u;
    return u;
}

bool unite(int u, int v, int w){
    int u_ = u, v_ = v;
    u = find(u); v = find(v);
    if (u == v) return false;
    if (uf[u] > uf[v]) u^=v^=u^=v;
    uf[u] += uf[v];
    uf[v] = u;
    add_edge(u_, v_, w);
    return true;
}

void kruskal(int n, int m){
    sort(ed+1, ed+1+m);
    int cnt = 0;
    _rfor(i, 1, m){
        cnt += unite(ed[i].u, ed[i].v, ed[i].w);
        if (cnt == n-1) break;
    }
    if (cnt < n-1){
        int u = 0;
        _rfor(i, 1, n){
            if (uf[i] > 0) continue;
            if (u) cnt += unite(u, i, inf);
            u = i;
        }
        assert(cnt == n-1);
    }
}
```

Kruskal 重构树

```
#define maxn 100005
#define maxm 200005
#define lbmaxn 21
struct Edge{
    int u, v, w;
    inline bool operator<(const Edge &t)const{ return w < t.w; }
} ed[maxm];

int ecnt = 1, ncnt, lbn,
    head[maxn],
    to[maxn*2],
    nex[maxn*2],
    we[maxn*2],
    ch[maxn*2][2],
    vl[maxn*2],
    fa[lbmaxn][maxn*2],
    uf[maxn],
    uu[maxn],
    stk[maxn];

void add_edge(int u, int v, int w){
    to[++ecnt] = v; nex[ecnt] = head[u]; head[u] = ecnt; we[ecnt] = w;
    to[++ecnt] = u; nex[ecnt] = head[v]; head[v] = ecnt; we[ecnt] = w;
}

int find(int u){
    int top = 0;
    while(uf[u] > 0) stk[top++] = u, u = uf[u];
    while(top > 0) uf[stk[--top]] = u;
    return u;
}

bool unite(int u, int v, int w){
    int u_ = u, v_ = v;
    u = find(u); v = find(v);
    if (u == v) return false;
    if (uf[u] > uf[v]) u^=v^=u^=v;
    uf[u] += uf[v];
    uf[v] = u;
    add_edge(u_, v_, w);
    fa[0][uu[u]] = fa[0][uu[v]] = ++ncnt;
    ch[ncnt][0] = uu[u]; ch[ncnt][1] = uu[v];
    vl[ncnt] = w;
    uu[u] = ncnt;
    return true;
}
```

```
void dfs(int u){
    _rfor(i, 1, lbn) fa[i][u] = fa[i-1][fa[i-1][u]];
    if (vl[u] == 0) return;
    dfs(ch[u][0]); dfs(ch[u][1]);
}

void kruskal(int n, int m){
    ncnt = n;
    _rfor(i, 1, n) uf[i] = -1, uu[i] = i, vl[i] = 0;
    _rfor(i, 1, m){
        unite(ed[i].u, ed[i].v, ed[i].w);
        if (ncnt == 2*n-1) break;
    }
    if (ncnt < 2*n-1){
        int u = 0;
        _rfor(i, 1, n){
            if (uf[i] > 0) continue;
            if (u) unite(u, i, inf);
            u = i;
        }
    }
    int tmp = ncnt; lbn = 0;
    while(tmp) ++lbn, tmp >>= 1;
    fa[0][ncnt] = ncnt;
    dfs(ncnt);
}

int get_top(int u, int w){
    for(int i = lbn; i >= 0; --i) if (vl[fa[i][u]] <= w) u = fa[i][u];
    return u;
}
```

Boruvka

```
#define maxn 100005
#define maxm 200005
struct Edge{ int u, v, w; } ed[maxm];
int ecnt = 1,
    head[maxn],
    to[maxn*2],
    nex[maxn*2],
    we[maxn*2],
    uf[maxn],
    stk[maxn],
    key[maxn];
bool del_tag[maxn];

void add_edge(int u, int v, int w){
    to[++ecnt] = v; nex[ecnt] = head[u]; head[u] = ecnt; we[ecnt] = w;
    to[++ecnt] = u; nex[ecnt] = head[v]; head[v] = ecnt; we[ecnt] = w;
}

int find(int u){
    int top = 0;
    while(uf[u] > 0) stk[top++] = u, u = uf[u];
    while(top > 0) uf[stk[--top]] = u;
    return u;
}
```

```
bool unite(int u, int v, int w){
    int u_ = u, v_ = v;
    u = find(u); v = find(v);
    if (u == v) return false;
    if (uf[u] > uf[v]) u^=v^=u^=v;
    uf[u] += uf[v];
    uf[v] = u;
    add_edge(u_, v_, w);
    return true;
}

void boruvka(int n, int m){
    ms(uf, -1);
    int cnt = 0;
    while(cnt != n-1){
        _rfor(i, 1, n) key[i] = 0;
        bool flag = false;
        _rfor(i, 1, m){
            if (del_tag[i]) continue;
            int u = ed[i].u, v = ed[i].v, w = ed[i].w;
            if ((u = find(u)) == (v = find(v))){ del_tag[i] = true; continue; }
            if (!key[u] || ed[key[u]].w < w) key[u] = i;
            if (!key[v] || ed[key[v]].w < w) key[v] = i;
            flag = true;
        }
        if (!flag) break; //整个图不连通
        _rfor(i, 1, n) if (uf[i] < 0 && key[i]) cnt += unite(ed[key[i]].u, ed[key[i]].v, ed[key[i]].w);
    }
    if (cnt < n-1){
        int u = 0;
        _rfor(i, 1, n){
            if (uf[i] > 0) continue;
            if (u) cnt += unite(u, i, inf);
            u = i;
        }
        assert(cnt == n-1);
    }
}
```

ISAP

```
#define maxn /*...*/
#define maxm /*...*/
struct NetworkFlow{
    struct Node{ int head, cur, g, h; } node[maxn];
    struct Edge{ int to, nex, cap; } edge[maxm];
    #define head(x) node[x].head
    #define cur(x) node[x].cur
    #define g(x) node[x].g
    #define h(x) node[x].h
    #define to(x) edge[x].to
    #define nex(x) edge[x].nex
    #define cap(x) edge[x].cap
    int n, s, t, ecnt, max_flow, que[maxn];
    void init(int n_){ n = n_; ecnt = s = 1; t = 2; _rfor(i, 1, n) head(i) = 0; }
    void ae(int u, int v, int c){
        to(++ecnt) = v; nex(ecnt) = head(u); head(u) = ecnt; cap(ecnt) = c;
        to(++ecnt) = u; nex(ecnt) = head(v); head(v) = ecnt; cap(ecnt) = 0;
    }
    void set_h(){
        re int fro = 0, bac = 0;
        _rfor(i, 1, n) h(i) = g(i) = 0;
        g(h(que[bac++] = t) = 1) = 1;
        while(bac > fro){
            re int u = que[fro++], v;
            _fev(p, u) if (!h(v = to(p))) ++g(h(que[bac++] = v) = h(u)+1);
        }
        if (!h(s)) h(s) = n+1;
    }
    int dfs(int u, int flow){
        if(u == t){ max_flow += flow; return flow; }
        int used = 0, tmp;
        for(re int p = cur(u); p; p = nex(p)){
            cur(u) = p;
            if (cap(p) == 0 || h(to(p)) != h(u) - 1) continue;
            tmp = dfs(to(p), min(cap(p), flow-used));
            if (tmp) cap(p) -= tmp, cap(p^1) += tmp, used += tmp;
            if (used == flow) return flow;
        }
        if (--g(h(u)) == 0) h(s) = n+1;
        ++g(++h(u));
        return used;
    }
    void isap(){
        max_flow = 0;
        set_h();
        while(h(s) <= n){
            _rfor(i, 1, n) cur(i) = head(i);
            dfs(s, inf);
        }
    }
    #undef head
    #undef cur
    #undef g
    #undef h
    #undef to
    #undef nex
    #undef cap
} nf;
```

Dinic

```
#define maxm /*...*/
#define maxn /*...*/
struct NetworkFlow{
    struct Node{ int head, cur, h; } node[maxn];
    struct Edge{ int to, nex, cap; } edge[maxm];
    #define head(x) node[x].head
    #define cur(x) node[x].cur
    #define h(x) node[x].h
    #define to(x) edge[x].to
    #define nex(x) edge[x].nex
    #define cap(x) edge[x].cap
    int n, s, t, ecnt, max_flow, que[maxn];
    void init(int n_){ n = n_; ecnt = s = 1; t = 2; _rfor(i, 1, n) head(i) = 0; }
    void ae(int u, int v, int c){
        to(++ecnt) = v; nex(ecnt) = head(u); head(u) = ecnt; cap(ecnt) = c;
        to(++ecnt) = u; nex(ecnt) = head(v); head(v) = ecnt; cap(ecnt) = 0;
    }
    bool set_h(){
        re int fro = 0, bac = 0;
        _rfor(i, 1, n) h(i) = 0;
        h(que[bac++] = s) = 1;
        while(bac > fro){
            re int u = que[fro++], v;
            _fev(p, u) if (!h(v = to(p)) && cap(p) > 0) h(que[bac++] = v) = h(u)+1;
        }
        return h(t) > 0;
    }
    int dfs(int u, int flow){
        if(u == t){ max_flow += flow; return flow; }
        int used = 0, tmp;
        for(re int p = cur(u); p; p = nex(p)){
            cur(u) = p;
            if (cap(p) == 0 || h(to(p)) != h(u) + 1) continue;
            tmp = dfs(to(p), min(cap(p), flow-used));
            if (tmp < min(cap(p), flow-used)) h(to(p)) = 0;
            if (tmp) cap(p) -= tmp, cap(p^1) += tmp, used += tmp;
            if (used == flow) return flow;
        }
        return used;
    }
    void dinic(){
        max_flow = 0;
        while(set_h()){
            _rfor(i, 1, n) cur(i) = head(i);
            dfs(s, inf);
        }
    }
    #undef head
    #undef cur
    #undef g
    #undef h
    #undef to
    #undef nex
    #undef cap
} nf;
```

匈牙利算法

```
int ecnt = 1, head[maxn], to[maxm*2], nex[maxm*2];
bool vis[maxn];
void dfs(int u){
    vis[u] = true;
    for(int p = head[u]; p; p = nex[p]){
        int v = to[p];
        if (vis[v]) continue;
        vis[v] = true;
        if (!mch[v] || (!vis[mch[v]] && dfs(mch[v]))){
            mch[mch[u] = v] = u;
            return true;
        }
    }
}
```

```
int match(int n1, int n2){
    int n = n1+n2;
    _rfor(i, 1, n1){
        if (mch[i]) continue;
        _rfor(j, 1, n) vis[j] = false;
        if (dfs(i)) ++ans;
    }
}
```

带花树算法

```
#define maxgn /*...*/
#define maxgn /*...*/
struct Blossom{
    int ncnt = 1, ecnt = 1, ccnt, fro, bac,
        head[maxgn], to[maxgm], nex[maxgm],
        mch[maxgn], pre[maxgn], col[maxgn], vis[maxgn], fa[maxgn],
        que[maxgn];
    void init(int n){ ecnt = 1; ncnt = n; _rfor(i, 1, n) head[i] = 0; }
    void add_edge(int u, int v){
        to[++ecnt] = v; nex[ecnt] = head[u]; head[u] = ecnt;
        to[++ecnt] = u; nex[ecnt] = head[v]; head[v] = ecnt;
    }

    void aug(int v){ for(int u, t; v; v = t) t = mch[u = pre[v]], mch[mch[u] = v] = u;
}
    int get_fa(int u){ return u == fa[u] ? u : get_fa(fa[u]); }

    int get_lca(int u, int v){
        ++ccnt;
        u = get_fa(u), v = get_fa(v);
        while(vis[u] != ccnt){
            vis[u] = ccnt;
            u = get_fa(pre[mch[u]]);
            if (v) u^=v^=u^=v;
        }
        return u;
    }

    void shrink(int u, int v, int lca){
        while(get_fa(u) != lca){
            pre[u] = v;
            v = mch[u];
            if (col[v] == 2) col[que[bac++] = v] = 1;
            fa[u] = fa[v] = lca;
            u = pre[v];
        }
    }

    bool bfs(int s){
        fro = bac = ccnt = 0;
        //清空步骤一定不能少//
        _rfor(i, 1, ncnt) pre[i] = col[i] = vis[i] = 0, fa[i] = i;
        col[que[bac++] = s] = 1;
        while(bac > fro){
            int u = que[fro++];
            for(int v, p = head[u]; p; p = nex[p]){
                if (!col[v = to[p]]){
                    pre[v] = u;
                    if (!mch[v]) return aug(v), true;
                    else col[v] = 2, col[que[bac++] = mch[v]] = 1;
                }
                else if (col[v] == 1 && get_fa(u) != get_fa(v)){
                    int lca = get_lca(u, v);
                    shrink(u, v, lca);
                    shrink(v, u, lca);
                }
            }
        }
        return false;
    }

    int blossom(){
        int pch = 0;
        _rfor(i, 1, ncnt) mch[i] = 0;
        _rfor(u, 1, ncnt) if (!mch[u])
            for(int v, p = head[u]; p; p = nex[p])
                //别忘了 break 出去//
                if (!mch[v = to[p]]){ ++pch, mch[mch[u] = v] = u; break; }
        _rfor(i, 1, ncnt) if (!mch[i]) pch += bfs(i);
        return pch;
    }
} bls;
```

费用流

```
struct Graph{
    typedef int gtype;
    int ecnt, ncnt, s, t,
        head[maxn],
        to[maxm],
        nex[maxm],
        pre[maxn],
        que[2*maxn],
        cnt[maxn];
    gtype
        mf, mc,
        dis[maxn],
        cap[maxm],
        cst[maxm];
    bool inque[2*maxn];
    void init(int n){ s = ecnt = 1; t = 2; ncnt = n; ms(head, 0); }
    void add_edge(int u, int v, gtype cp, gtype cs){
        to[++ecnt] = v; nex[ecnt] = head[u]; head[u] = ecnt; cap[ecnt] = cp; cst[ecnt]
= cs;
        to[++ecnt] = u; nex[ecnt] = head[v]; head[v] = ecnt; cap[ecnt] = 0; cst[ecnt] =
-cs;
    }
    bool spfa(){
        re int fro = 0, bac = 0, u, v; ms(pre, 0); ms(cnt, 0); ms(inque, false);
        _rfor(i, 1, ncnt) dis[i] = inf;
        dis[s] = 0; inque[que[bac++] = s] = true;
        while(bac != fro){
            inque[u = que[fro++]] = false; fro %= 2*maxn;
            _fev(p, u){
                if (cap[p] && dis[u] + cst[p] < dis[v = to[p]]){
                    dis[v] = dis[u] + cst[p];
                    pre[v] = p;
                    if (!inque[v])
                        inque[que[bac++] = v] = true, bac %= 2*maxn, cnt[v]++;
                    if (cnt[v] > ncnt){
                        pf("Wrong in Negative Circle!\n");
                        return false;
                    }
                }
            }
        }
        return dis[t] < inf;
    }
    void augment(){
        gtype af = inf;
        int p, u;
        for(p = pre[u = t]; u != s; p = pre[u])
            af = min(af, cap[p]), u = to[p^1];
        for(p = pre[u = t]; u != s; p = pre[u])
            cap[p] -= af, cap[p^1] += af, u = to[p^1];
        mf += af; mc += af*dis[t];
    }
    void min_cost_max_flow(){
        mf = 0; mc = 0;
        while(spfa()) augment();
    }
    void min_cost(){
        mf = 0; mc = 0;
        while(spfa() && dis[t] < 0) augment();
    }
} g;
```

数论

快速幂

```
int fp(int x, int n, int p){
    int y = 1;
    for(; n; x = (ll)x*x%p, n >>= 1) if (n&1) y = (ll)x*y%p;
    return y;
}
```

快速乘

```
ll mul(ll x, ll y, ll p){
    return ((x*y-(ll)((long double)x/p*y)*p)%p+p)%p;
}
```

光速幂（块速幂）

```
#define maxs 320
int base, p, sqr,
    quo[maxs],
    rem[maxs];

void prepare(int n){
    sqr = sqrt(n+0.5);
    quo[0] = rem[0] = 1;
    _rfor(i, 1, sqr) rem[i] = (ll)rem[i-1]*base%p;
    quo[1] = rem[sqr];
    _rfor(i, 2, n/sqr) quo[i] = (ll)quo[i-1]*quo[1]%p;
}

int fp(int n, int p){
    return (ll)quo[n/sqr]*rem[n%sqr]%p;
}
```

快速取模

```
#define ull unsigned long long
struct FastMod{
    ull p, m;
    FastMod(ull p):p(p), m(((__uint128_t)1<<64)/p){}
    friend inline ull operator%(const ull&a, const FastMod &mod){
        re ull r = a-(((__uint128_t)mod.m*a)>>64)*mod.p;
        return r>=mod.p ? r-mod.p : r;
    }
} mod(998244353);
```

逆元

```
typedef int type;

type gcd(type a, type b){
    while(b) a %= b, a^=b^=a^=b;
    return a;
}

void exgcd(type a, type b, type &g, type &x, type &y){
    if (!b) g = a, x = 1, y = 0;
    else exgcd(b, a%b, g, y, x), y -= x*(a/b);
}

//扩展欧几里得求逆元
type get_inv(type a, type p){
    int x, y, g;
    exgcd(p, a%p, g, x, y);
    if (g != 1) return -1;
    return (y%p+p)%p;
}

#define mod /*...*/
type fp(type x, type n){
    type y = 1;
    for(; n; x = (ll)x*x%mod, n>>=1) if (n&1) y = (ll)x*y%mod;
    return y;
}

//费马小定理求逆元
int get_inv(int x){
    return fp(x, mod-2);
}

//逆元的线性递推
```

```
//因为 [mod%i] 不好 cache，所以线性递推不如线性筛快
void init(int n){
    inv[1] = 1;
    _rfor(i, 2, n) inv[i] = (ll)(mod - mod/i)*inv[mod%i]%mod;
}
```

埃氏筛

```
#define maxn 1000005
int pcnt = 0,
    phi[maxn],
    prime[maxn];
bool vis[maxn];

//求素数
void get_prime(int n){
    _rfor(i, 2, n){
        if (vis[i]) continue;
        prime[++pcnt] = i;
        if (i > sqrt(n)) continue;
        for(int j = i*i; j <= n; j += i) vis[j] = true;
    }
}
```

线性筛（欧拉筛）

```
#define maxn 1000005
int pcnt = 0,
    mu[maxn],
    phi[maxn],
    prime[maxn];
bool vis[maxn];

void linear_sieve(int n){
    phi[1] = mu[1] = 1;
    _rfor(i, 2, n){
        if (!vis[i]) prime[++pcnt] = i, phi[i] = i-1, mu[i] = -1;
        for(int j = 1; j <= pcnt && i*prime[j] <= n; ++j){
            vis[i*prime[j]] = true;
            if (i % prime[j] == 0){ phi[i*prime[j]] = prime[j]*phi[i], mu[i*prime[j]] =
0; break; }
            else phi[i] = (prime[j]-1)*phi[i], mu[i*prime[j]] = mu[i]*-1;
        }
    }
}
```

杜教筛

```
int lim, f[maxn];
unordered_map<ll, int> sumfmp;
void init(int n){
    /*...To get f[]...*/
    _rfor(i, 1, n) f[i] += f[i-1];
}

int sumh(ll n);
//phi:sumh(n)=n%mod*(n+1)%mod*(mod-mod/2)%mod;
```

```
//mu:sumh(n)=n%mod
int sumg(ll n);
//phi:sumg(n)=n%mod
//mu:sumg(n)=n/%mod

int sumf(ll n){
    if (n <= lim) return f[(int)n];
    else if (sumfmp.count(n)) return sumfmp[n];
    else{
        int ans = 0;
        for(ll l = 2, r; l <= n; l = r+1){
            r = n/(n/l);
            ans = (ans+(sumg(r)+mod-sumg(l-1))*sumf(n/l))%mod;
        }
        return mp[n] = (sumh(n)+mod-ans)%mod;
    }
}
```

快速素性判断

```
ll  test_gcd = (ll)2*3*5*7*11*13*17*19*23*29*31*37*41*43;
int tcnt = 8, test_prime[] = {0, 2, 3, 5, 7, 11, 13, 17, 19};

ll mul(ll x, ll y, ll mod){ return ((x*y-(ll)((long double)x/mod*y)*mod)%mod+mod)%mod;
}
ll fp(ll x, ll n, ll mod){
    ll y=1;
    for(; n; x = mul(x, x, mod), n >>= 1) if (n&1) y = mul(x, y, mod);
    return y;
}

ll gcd(ll a, ll b){
    while(b) a %= b, a^=b^=a^=b;
    return a;
}

bool miller_rabin(ll a, ll p){
    ll x = p-1, y;
    if (fp(a, x, p) != 1) return false;
    do{
        y = fp(a, x>>=1, p);
        if (y == p-1) return true;
        if (y != 1) return false;
    }while(!(x&1));
    return true;
}

bool mr(ll a, ll p){
    int t = lb((p-1)&(1-p));
    ll t1 = fp(a, (p-1)>>t, p);
    if (t1 == 1) return true;
    for(ll t2; t; --t, t1 = t2){
        t2 = mul(t1, t1, p);
        if (t2 == 1) return t1 == p-1;
    }
    return false;
}

bool is_prime(ll n){
    if (n % 6 != 1 && n % 6 != 5) return false;
    if (gcd(test_gcd, n) != 1) return false;
    _rfor(i, 1, tcnt) if (!miller_rabin(test_prime[i], n)) return false;
    return true;
}
```

exBSGS

```
int fp(int x, int n, int p){
    int y = 1;
    for(; n; x = (ll)x*x%p, n >>= 1) if (n&1) y = (ll)x*y%p;
    return y;
}

int bsgs(int a, int b, int p){
    if (b == 1) return 0;
    unordered_map<int, int> mp;
    int t = sqrt(p+0.5)+1, at = fp(a, t, p);
    _for(i, 0, t) mp[b] = i, b = (ll)b*a%p;
    a = at;
    _rfor(i, 1, t){
        if (mp.cound(a)) return i*t-mp[a];
        a = (ll)a*at%p;
    }
    return -1;
}

int gcd(int a, int b){
    while(b) a %= b, a^=b^=a^=b;
    return a;
}

void exgcd(int a, int b, int &g, int &x, int &y){
    if (!b) g = a, x = 1, y = 0;
    else exgcd(b, a%b, g, y, x), y -= x*(a/b);
}

int get_inv(int a, int p){
    int g, x, y;
    exgcd(a, p, g, x, y);
    if (g != 1) return -1;
    return (x%p+p)%p;
}

int exbsgs(int a, int b, int p){
    if (b == 1) return 0;
    if (gcd(a, p) == 1) return bsgs(a, b, p);
    int d = 1, k = 0, g;
    while((g = gcd(a, p)) != 1){
        if (b % g != 0) return -1;
        ++k; b /= g; p /= g; d = (ll)d*(a/g)%p;
        if (d == b) return k;
    }
    int x = bsgs(a, (ll)b*get_inv(d, p)%p, p);
    return (x >= 0) ? x+k : -1;
}
```

二次剩余

```
#include <random>

mt19937 engine(315);
uniform_int_distribution<ll> dstr(0, 0x7fffffffffffffffll);
ll randnt(){ return dstr(engine); }

struct Cplx{
    static int w, p; int x, y;
    Cplx operator*(const Cplx &t)const{ return {((ll)x*t.x+(ll)w*y%p*t.y)%p,
((ll)x*t.y+(ll)y*t.x)%p}; }
    Cplx operator+(const Cplx &t)const{ return {((ll)x+t.x)%p, ((ll)y+t.y)%p}; }
    Cplx operator-(const Cplx &t)const{ return {((ll)x+p-t.x)%p, ((ll)y+p-t.y)%p}; }
    Cplx operator-()const{ return {x?p-x:0, y?p-y:0}; }
};
int Cplx::w, Cplx::p;

int fp(int x, int n, int p){
    int y = 1;
    for(; n; x = (ll)x*x%p, n >>= 1) if (n&1) y = (ll)x*y%p;
    return y;
}

Cplx fp(Cplx x, int n){
    Cplx y = {1, 0};
    for(; n; x = x*x, n >>= 1) if (n&1) y = x*y;
    return y;
}

int quandratic(int a, int p){
    if (a == 0) return 0;
    if (p <= 2) return a%p;
    if (fp(a, (p-1)/2, p) != 1) return -1;
    re int b; a %= Cplx::p = p;
    do{ b = randnt()%p; }while(fp(Cplx::w=((ll)b*b%p+p-a)%p, (p-1)/2, p) == 1);
    Cplx as = fp((Cplx){b, 1}, (p+1)/2); assert(as.y == 0);
    return as.x*2 < p ? as.x : p-as.x;
}
```

CRT

```
ll mul(re int ll a, re ll b, re ll m){ return ((a*b-(ll)((long double)a*b/m)*m)%m+m)%m;
}
ll gcd(re ll a, re ll b){ while(b) a%=b, a^=b^=a^=b; return a; }
void exgcd(ll a, ll b, ll &g, ll &x, ll &y);
ll get_inv(re ll a, re ll p);

struct CRT{
    int a[maxn], m[maxn];
    ll solve(re int n){
        re ll M = 1, x = 0;
        _rfor(i, 1, n) M *= m[i];
        _rfor(i, 1, n) x = (x+mul(get_inv(M/m[i]%m[i], m[i]), mul(M/m[i], a[i], M),
M))%M;
        return x;
    }
} crt;
```

exCRT

```
ll mul(re ll a, re ll b, re ll m){ return ((a*b-(ll)((long double)a*b/m)*m)%m+m)%m; }
ll gcd(re ll a, re ll b){ while(b) a%=b, a^=b^=a^=b; return a; }
void exgcd(ll a, ll b, ll &g, ll &x, ll &y);
ll get_inv(re ll a, re ll p);

struct exCRT{
    ll a[maxn], m[maxn];
    ll solve(re int n){
        re ll M = m[1], x = a[1];
        _rfor(i, 2, n){
            re ll g = gcd(M, m[i]), mm = M/g*m[i];
            if (x % m[i] == a[i]){ M = mm; continue; }
            if ((x-a[i])%g) return -1;
            x = mul(mul(get_inv(m[i]/g, M/g), m[i], mm), (x>=a[i]?(x-a[i])/g:mm+(x-
a[i])/g), mm)+a[i];
            if (x >= mm) x -= mm; M = mm;
        }
        return x;
    }
} crt;
```

exLucas

```
void exgcd(ll a, ll b, ll &g, ll &x, ll &y);
ll get_inv(re ll a, re ll p);
ll fp(int x, int n, int p);

struct exLucas{
    int f(ll n, int p, int m){
        if (!n) return 1;
        int ans = 1;
        _rfor(i, 2, m) if (i%p) ans = (ll)ans*i%m;
        ans = fp(ans, n/m, m);
        _rfor(i, 2, (int)(n%m)) if (i%p) ans = (ll)ans*i%m;
        return (ll)ans*f(n/p, p, m)%m;
    }
    ll g(ll n, int p){
        ll ans = 0;
        while(n >= p) ans += n/p, n /= p;
        return ans;
    }
    ll C(ll n, ll m, int p, int e){
        ll x = g(n, p)-g(m, p)-g(n-m, p);
        if (x >= e) return 0;
        int pe = fp(p, e, 1000000000);
        return (ll)f(n, p, pe)*get_inv(f(m, p, pe), pe)%pe*get_inv(f(n-m, p, pe),
pe)%pe*fp(p, x, pe)%pe;
    }
} lucas;
```

Pollard-Rho

```
mt19937 engine(23423);
uniform_int_distribution<ll> dstr(0, 0x7ffffffffffffffllu);
ll randnt(){ return dstr(engine); }
ll mul(ll a, ll b, ll p){long long r=a*b-(long long)((long double)a*b/p+0.5)*p;return r>=0?r:r+p;}
ll fp(ll x, ll n, ll p){ll y=1;for(;n;x=mul(x,x,p),n>>=1)if(n&1)y=mul(x,y,p);return y;}
ll gcd(ll a, ll b){ while(b) a %= b, swap(a, b); return a; }
int lb(ll x){ return x ? 63-__builtin_clzll(x) : -1; }
bool is_prime(ll n);

ll  fcnt = 0, fct[100];

ll get_fct(ll n){
    //_rfor(i, 1, 100) if (n%prime[i] == 0) return prime[i];
    ll x1 = randnt()%n, x2 = x1, c = randnt()%n, res = 1;
    x2 = (mul(x2, x2, n)+c)%n;
    int step = lb(n);
    for(re int i = 0; x1 != x2; ++i){
        res = mul(res, x2-x1+n, n);
        if (!res) res = x2-x1+n;
        if (i == step){
            ll g = gcd(res, n);
            if (g != 1) return g;
            res = 1;
            i = 0;
        }
        x1 = (mul(x1, x1, n)+c)%n;
        x2 = (mul(x2, x2, n)+c)%n;
        x2 = (mul(x2, x2, n)+c)%n;
    }
    return gcd(res, n);
}

void prho(ll n){
    //assert(!is_prime(n));
    ll k = 1;
    while(k == 1 || k == n) k = get_fct(n);
    n /= k;
    static bool fg; fg = false;
    if (is_prime(k) || (fg = true, swap(k, n), is_prime(k))){
        fct[++fcnt] = k;
        if (n%k == 0){ do{ n /= k, fct[++fcnt] = k; }while(n%k == 0); }
        else if (fg) return prho(n);
        if (n == 1) return;
        if (is_prime(n)){ fct[++fcnt] = n; return; }
        else prho(n);
    }
    else prho(k), prho(n);
}
```

多项式

FFT

```
struct Cplx{
    db x, y;
    Cplx(db x = 0.0, db y = 0.0):x(x), y(y){}
    Cplx(const Cplx &t):x(t.x), y(t.y){}
    Cplx &operator=(const Cplx &t){ x = t.x; y = t.y; return *this; }
    Cplx operator+(const Cplx &t)const{ return {x+t.x, y+t.y}; }
    Cplx operator-(const Cplx &t)const{ return {x-t.x, y-t.y}; }
    Cplx operator-()const{ return {-x, -y}; }
    Cplx operator*(db n)const{ return {x*n, y*n}; }
    Cplx operator*(const Cplx &t)const{ return {x*t.x-y*t.y, x*t.y+t.x*y}; }
    Cplx operator/(db n)const{ return {x/n, y/n}; }
    Cplx operator/(const Cplx &t)const{ return Cplx(x*t.x+y*t.y, t.x*y-
x*t.y)/(t.x*t.x+t.y*t.y); }
    Cplx conj()const{ return {x, -y}; }
};

const db tao = 2.0*acos(-1);

int lb(int n){ int lbn = -1; while(n) n>>=1, ++lbn; return lbn; }

Cplx omg(int n, int k){ return {cos(tao*k/n), sin(tao*k/n)}; }

void fft(Cplx *a, int n, bool fg = false){
    assert(n == (n&-n));
    static Cplx w[maxn]; static int tr[maxn];
    int lbn = lb(n); w[0] = {1, 0}; tr[0] = 0;
    _for(i, 1, n) tr[i] = (tr[i>>1]>>1)|((i&1)<<lbn-1);
    _for(i, 0, n) if (i < tr[i]) swap(a[i], a[tr[i]]);
    for(re int b = 1; b < n; b <<= 1){
        _for(k, 1, b) w[k] = omg(2*b, fg?-k:k);
        for(Cplx *p = a, t; p < a+n; p += 2*b) _for(k, 0, b)
            t = w[k]*p[b|k], p[b|k] = p[k]-t, p[k] = p[k]+t;
    }
    if (fg) _for(i, 0, n) a[i] = a[i]/n;
}

#define ifft(a, n) fft(a, n, true)

void dbfft(Cplx *a, Cplx *b, int n){
    assert(n == (n&-n));
    static Cplx p[maxn];
    _for(i, 0, n) p[i] = {a[i].x, b[i].x};
    fft(p, n);
    a[0] = {p[0].x, 0.0}; b[0] = {p[0].y, 0.0};
    _for(i, 1, n) a[i] = (p[i]+p[n-i].conj())*0.5, b[i] = (p[i]-p[n-i].conj())*Cplx(0,
-0.5);
}

struct Poly{
    Cplx a[maxn]; int n;
    Poly &operator=(const Poly &t){ n = t.n; _for(i, 0, n) a[i] = t.a[i]; return *this;
}
    Poly &resize(int nn){ _for(i, n, nn) a[i] = {0.0, 0.0}; n = nn; return *this; }
    Poly &real_mul(const Poly &t){
        int nn = 1<<(lb(max(n, t.n)-1)+1);
        static Cplx a0[maxn], a1[maxn], b0[maxn], b1[maxn], p[maxn];
        _for(i, 0, nn/2){
            a0[i] = {2*i<n?a[2*i].x:0.0, 0.0}, a1[i] = {2*i+1<n?a[2*i+1].x:0.0, 0.0};
            b0[i] = {2*i<t.n?t.a[2*i].x:0.0, 0.0}, b1[i] = {2*i+1<t.n?t.a[2*i+1].x:0.0,
0.0};
        }
        _for(i, nn/2, nn) a0[i] = a1[i] = b0[i] = b1[i] = {0.0, 0.0};
        dbfft(a0, b0, nn); dbfft(a1, b1, nn);
        _for(i, 0, nn)
            p[i] = (a0[i]*b0[i]+omg(nn,i)*a1[i]*b1[i])+
(a1[i]*b0[i]+a0[i]*b1[i])*Cplx(0, 1);
```

```
        ifft(p, nn);
        _for(i, 0, nn) a[2*i] = {p[i].x, 0}, a[2*i+1] = {p[i].y, 0}; n = 2*nn;
        return *this;
    }
    Poly &mul(const Poly &t){
        int nn = 1<<(lb(n+t.n-2)+1);
        static Cplx b[maxn];
        _for(i, 0, nn) b[i] = i<t.n?t.a[i]:Cplx(0.0,0.0);
        _for(i, n, nn) a[i] = {0.0, 0.0};
        fft(a, nn); fft(b, nn);
        _for(i, 0, nn) a[i] = a[i]*b[i];
        ifft(a, nn); n = nn;
        return *this;
    }
};
```

MTT

```
#define msk 32767

void exgcd(int a, int b, int &g, int &x, int &y);
int get_inv(int a, int p);
int lb(int n);
struct Cplx;
Cplx omg(int n, int k);
void fft(Cplx *a, int n, bool fg = false);
#define ifft(a, n) fft(a, n, true)
void dbfft(Cplx *a, Cplx *b, int n);
#define turn(x) ((ll)(x+0.5))

struct MTT{
    static int p;
    int a[maxn], n;
    static void init(int p_){ p = p_; }
    MTT &operator=(const MTT &t){ n = t.n; _for(i, 0, n) a[i] = t.a[i]; return *this; }
    MTT &resize(re int nn){ _for(i, n, nn) a[i] = 0; n = nn; return *this; }
    MTT &mul(const MTT &t){
        int nn = 1<<lb(n+t.n-2)+1;
        static Cplx a0[maxn], a1[maxn], b0[maxn], b1[maxn], c[maxn];
        _for(i, 0, nn){
            a0[i] = {i<n?a[i]&msk:0.0, 0.0}, a1[i] = {i<n?a[i]>>15:0.0, 0.0};
            b0[i] = {i<t.n?t.a[i]&msk:0.0, 0.0}, b1[i] = {i<t.n?t.a[i]>>15:0.0, 0.0};
        }
        dbfft(a0, a1, nn); dbfft(b0, b1, nn);
        _for(i, 0, nn) c[i] = a0[i]*b0[i]+a1[i]*b1[i]*Cplx(0, 1);
        _for(i, 0, nn) a0[i] = a0[i]*b1[i]+a1[i]*b0[i];
        ifft(c, nn); ifft(a0, nn);
        _for(i, 0, nn) a[i] = ((turn(c[i].y)%p<<30)+
(turn(a0[i].x)%p<<15)+turn(c[i].x))%p;
        n = nn;
        return *this;
    }
};
int MTT::p;
```

NTT

```
mt19937 engine(315);
uniform_int_distribution<ll> dstr(0, 0x7fffffffffffffffll);
ll randnt(){ return dstr(engine); }

struct Cplx{
    static int w; int x, y;
    Cplx operator*(const Cplx &t)const{
        return {((ll)x*t.x+(ll)w*y%mod*t.y)%mod, ((ll)x*t.y+(ll)y*t.x)%mod};
    }
};
int Cplx::w;

int fp(re int x, re int n){ re int y=1; for(;n;x=(ll)x*x%mod,n>>=1)if(n&1)y=
(ll)x*y%mod; return y; }
int get_inv(re int a){ return fp(a, mod-2); }
int lb(re int n){ re int lbn = -1; while(n > 0) ++lbn, n>>=1; return lbn; }

Cplx fp(Cplx x, int n){
    Cplx y = {1, 0};
    for(; n; x = x*x, n >>= 1) if (n&1) y = x*y;
    return y;
}

int quandratic(int a){
    if (a == 0) return 0;
    if (fp(a, (mod-1)/2) != 1) return -1;
    re int b; a %= mod;
    do{ b = randnt()%mod; }while(fp(Cplx::w=((ll)b*b%mod+mod-a)%mod, (mod-1)/2) == 1);
    Cplx as = fp((Cplx){b, 1}, (mod+1)/2); assert(as.y == 0);
    return as.x*2 < mod ? as.x : mod-as.x;
}

int g, iv[maxn], fct[maxn], ivf[maxn];
void init_ntt(re int n){
    int fac[100], fcnt = 0, t = mod-1, sqr = sqrt(t+0.5);
    _rfor(x, 2, sqr){
        if (t == 1) break;
        if (t % x == 0){ fac[++fcnt] = x; while(t % x == 0) t /= x; }
    }
    if (t != 1) fac[++fcnt] = t;
    _rfor(x, 2, 1000){
        g = x;
        _rfor(i, 1, fcnt) if (fp(x, (mod-1)/fac[i]) == 1){ g = 0; break; }
        if (g) break;
    }
    assert(g != 0);
    iv[1] = ivf[1] = ivf[0] = fct[1] = fct[0] = 1;
    _rfor(i, 2, n){
        iv[i] = (ll)iv[mod%i]*(mod-mod/i)%mod;
        fct[i] = (ll)fct[i-1]*i%mod;
        ivf[i] = (ll)ivf[i-1]*iv[i]%mod;
    }
}

void ntt(re int *arr, re int n, re bool fg = false){
    assert(n == (n&-n));
    static ull a[maxn]; static int tr[maxn], w[maxn];
    re int lbn = lb(n); tr[0] = 0; w[0] = 1;
    _for(i, 1, n) tr[i] = (tr[i>>1]>>1)|((i&1)<<lbn-1);
    _for(i, 0, n) a[i] = arr[tr[i]];
    for(re int b = 1, t; b < n; b <<= 1){
        w[1] = fp(g, (mod-1)/(2*b));
        _for(k, 2, b) w[k] = (ll)w[k-1]*w[1]%mod;
        for(re ull *p = a, t; p < a + n; p += b<<1) _for(k, 0, b)
            t = p[b|k]*w[k]%mod, p[b|k] = p[k]+mod-t, p[k] += t;
        if (b == (1<<17)) _for(i, 0, n) a[i] %= mod;
    }
    if(fg){
        re ull d = get_inv(n); arr[0] = a[0]%mod*d%mod;
        _for(i, 1, n) arr[i] = a[n-i]%mod*d%mod;
    }
    else _for(i, 0, n) arr[i] = a[i]%mod;
}

#define intt(arr, n) ntt(arr, n, true)

struct Poly{
    int n, a[maxn];
    Poly &resize(re int nn){ _for(i, n, nn) a[i] = 0; n = nn; return *this; }
    Poly &operator=(const Poly &t){ n = t.n; _for(i, 0, n) a[i] = t.a[i]; return *this;
}
    Poly &mul(const Poly &t){ //3ntt(2n) = 6ntt(n)
        static int b[maxn];
        re int nn = 1<<lb(n+t.n-2)+1; resize(nn);
        _for(i, 0, nn) b[i] = i<t.n?t.a[i]:0;
        ntt(a, nn); ntt(b, nn);
        _for(i, 0, nn) a[i] = (ll)a[i]*b[i]%mod;
        intt(a, nn);
        return *this;
    }
    Poly &inv(){ //2*6ntt(n) = 12ntt(n)
        static int b[maxn], c[maxn];
        re int nn = 1<<lb(n-1)+1; resize(nn);
        b[0] = get_inv(a[0]);
        for(re int n = 2; n <= nn; n <<= 1){
            _for(i, n/2, n) b[i] = 0;
            _for(i, 0, n) c[i] = a[i];
            ntt(b, n); ntt(c, n);
            _for(i, 0, n) c[i] = (ll)b[i]*c[i]%mod;
            intt(c, n);
            _for(i, n/2, n) c[i-n/2] = c[i], c[i] = 0;
            ntt(c, n);
            _for(i, 0, n) c[i] = (ll)b[i]*c[i]%mod;
            intt(b, n); intt(c, n);
            _for(i, n/2, n) b[i] = (mod-c[i-n/2])%mod;
        }
        _for(i, 0, nn) a[i] = b[i];
        return *this;
    }
    bool sqr_success;
    Poly &sqr(){ //2*(inv(n)+mul(n)) = 36ntt(n)
        static Poly p, q;
        re int nn = 1<<lb(n-1)+1, iv2 = get_inv(2); resize(nn);
        q.a[0] = quandratic(a[0]); q.n = 1;
        if (q.a[0] == -1){ sqr_success = false; return *this; }
        for(re int n = 2; n <= nn; n <<= 1){
            this->n = n;
            (p = q).resize(n).inv().mul(*this);
            _for(i, n/2, n) q.a[i] = (ll)iv2*p.a[i]%mod; q.n = n;
        }
        sqr_success = true;
        return *this = q;
    }
    Poly &dif(){
        if (n) --n;
        _for(i, 0, n) a[i] = (ll)a[i+1]*(i+1)%mod;
        return *this;
    }
    Poly &ing(){
        for(re int i = n++; i >= 1; --i) a[i] = (ll)a[i-1]*iv[i]%mod; a[0] = 0;
        return *this;
    }
    bool log_success;
    Poly &log(){ //inv(n)+mul(n) = 18ntt(n)
        static Poly p; p = *this;
        if (a[0] != 1){ log_success = false; return *this; }
        re int tn = n;
        dif().mul(p.inv().resize(tn-1)).resize(tn-1).ing();
        log_success = true;
        return *this;
    }
    bool exp_success;
    Poly &exp(){ //2*(log(n)+mul(n/2)) = 42ntt(n)
        static Poly p, q;
        if (a[0] != 0){ exp_success = false; return *this; }
        re int nn = 1<<lb(n-1)+1; resize(nn);
        q.a[0] = 1; q.n = 1;
        for(re int n = 2; n <= nn; n <<= 1){
            (p = q).resize(n).log();
            _for(i, n/2, n) p.a[i-n/2] = (a[i]+mod-p.a[i])%mod;
            p.n = n/2; p.mul(q);
            _for(i, n/2, n) q.a[i] = p.a[i-n/2]; q.n = n;
        }
        exp_success = true;
        return *this = q;
    }
    Poly &rev(){ _for(i, 0, n/2) swap(a[i], a[n-1-i]); return *this; }
    Poly &div(const Poly &d, Poly &q, Poly &r){ //inv(n-d)+mul(n-d)+mul(d) = 12ntt(n-d)+6ntt(n)
        assert(d.n <= n);
        re int m = n-d.n+1;
        (q = d).rev().resize(m).inv().resize(m).mul((r = *this).rev().resize(m)).resize(m).rev();
        (r = q).resize(d.n-1).mul(d).resize(d.n-1);
        return *this;
    }
    Poly &left(re int m){
        if (m >= n) return resize(0);
        n = n-m;
        _for(i, 0, n) a[i] = a[i+m];
        return *this;
    }
    Poly &right(re int m){
        assert(n + m < maxn-5);
        n += m;
        for(re int i = n-1; i >= m; --i) a[i] = a[i-m];
        _for(i, 0, m) a[i] = 0;
        return *this;
    }
    Poly &fpp(re int k, re int k_, re int m){ //log(n)+exp(n) = 60ntt(n)
        re int a0, b = 0;
        n = min(n, m);
        while(b < n && !a[b]) ++b;
        if (b == n || (ll)k*b >= m) return resize(0);
        left(b); n = min(n, m-k*b);
        a0 = get_inv(a[0]);
        _for(i, 0, n) a[i] = (ll)a[i]*a0%mod;
        log().resize(n);
        assert(log_success);
        _for(i, 1, n) a[i] = (ll)a[i]*k%mod;
        exp().resize(m-k*b);
        assert(exp_success);
        a0 = fp(get_inv(a0), k_);
        _for(i, 0, n) a[i] = (ll)a[i]*a0%mod;
        return right(k*b);
    }
};

void test(){
    static Poly p, q;
    re int n = 128;
    init_ntt(1<<lb(n-1)+2);
    _for(i, 0, n) p.a[i] = 1+(ll)3*i*i%mod; p.n = n;
    q = p;
    ntt(p.a, n); ntt(p.a, n, true);
    _for(i, 0, n) assert(q.a[i] == p.a[i]);

    p.inv().inv();
    _for(i, 0, n) assert(q.a[i] == p.a[i]);

    p.sqr().mul(p).resize(n);
    _for(i, 0, n) assert(q.a[i] == p.a[i]);

    p.log().exp();
    _for(i, 0, n) assert(q.a[i] == p.a[i]);
}
```

FWT

```
//OR
void fwt(re int *arr, re int n, re bool fg = false){
    assert(n == (n&-n));
    static ll a[maxn];
    _for(i, 0, n) a[i] = arr[i];
    for(re int b = 1; b < n; b *= 2)
        for(re ll *p = a; p < a+n; p += 2*b) _for(k, 0, b)
            if (!fg) p[b|k] = p[b|k]+p[k];
            else p[b|k] = p[b|k]-p[k];
    _for(i, 0, n) arr[i] = a[i]%mod, arr[i]<0?arr[i]+=mod:0;
}
//AND
void fwt(re int *arr, re int n, re bool fg = false){
    assert(n == (n&-n));
    static ll a[maxn];
    _for(i, 0, n) a[i] = arr[i];
    for(re int b = 1; b < n; b *= 2)
```

```
        for(re ll *p = a; p < a+n; p += 2*b) _for(k, 0, b)
            if (!fg) p[k] = p[k]+p[b|k];
            else p[k] = p[k]-p[b|k];
    _for(i, 0, n) arr[i] = a[i]%mod, arr[i]<0?arr[i]+=mod:0;
}
//XOR
void fwt(re int *arr, re int n, re bool fg = false){
    assert(n == (n&-n));
    static ll a[maxn];
    _for(i, 0, n) a[i] = arr[i];
    for(re int b = 1; b < n; b *= 2)
        for(re ll *p = a, t; p < a+n; p += 2*b) _for(k, 0, b)
            if (!fg) t = p[k], p[k] = p[k]+p[b|k], p[b|k] = t-p[b|k];
            else t = p[k], p[k] = p[k]+p[b|k], p[b|k] = t-p[b|k];
    if (fg){
        re ll d = get_inv(n);
        _for(i, 0, n) arr[i] = (a[i]%mod+mod)*d%mod;
    }
    else _for(i, 0, n) arr[i] = a[i]%mod, arr[i]<0?arr[i]+=mod:0;
}
```

分治FFT

```
#define maxn
struct Poly{};

int f[maxn], g[maxn];

void domul(re int *f, re int *g, re int *h, re int n, re bool fg){
    static Poly p, q; p.n = q.n = n;
    _for(i, 0, n) p.a[i] = f[i], q.a[i] = g[i];
    if (fg){ _for(i, n, 2*n) q.a[i] = g[i]; q.n = 2*n; }
    p.mul(q).resize(2*n);
    _for(i, n, 2*n) h[i] = (h[i]+p.a[i])%mod;
}

void cdqntt(int l, int r){
    if (l == r){ g[i] = /*...*/ return; }
    int mi = (l+r)>>1, n = mi-l+1; assert(r-l+1 == 2*n);
    cdqntt(l, mi);
    domul(f+l, g, f+l, n, l >= 2*n);
    if (!l) domul(g+l, f, f+l, n, l >= 2*n);
}
```

一行第一类斯特林数

```
struct Poly{
    Poly &shift(re int v){
        static Poly p, q;
        re int tn = n;
        q.a[0] = 1; p.n = q.n = n;
        _for(i, 0, n) p.a[n-1-i] = (ll)fct[i]*a[i]%mod;
        _for(i, 1, n) q.a[i] = (ll)q.a[i-1]*v%mod*iv[i]%mod;
        p.mul(q).resize(n);
        _for(i, 0, n) a[i] = (ll)p.a[n-1-i]*ivf[i]%mod;
        return resize(tn);
    }
};
```

```
void upper(int n, Poly &p){
    static Poly q;
    if (n == 1){ p.a[0] = 0; p.a[1] = 1; p.n = 2; return; }
    upper(n/2, p); assert(p.n == n/2+1);
    p.mul((q = p).shift(n/2)).resize((n/2)*2+1);
    if (n&1){
        p.resize(n+1);
        for(re int i = n; i >= 1; --i)
            p.a[i] = (p.a[i-1]+(ll)(n-1)*p.a[i]%mod)%mod;
    }
}
```

线性代数

高斯消元

逆元版

```
int matrix[maxn][maxn];
bool gauss(int a[maxn][maxn], int n){
    _rfor(i, 1, n){
        int d, tar = 0;
        _rfor(j, i, n) if (a[j][i]){ tar = j; break; }
        if (!tar) return false;
        if (tar != i) _rfor(j, i, n+1) swap(a[tar][j], a[i][j]);
        d = get_inv(a[i][i]); a[i][i] = 1;
        _rfor(j, i+1, n+1) a[i][j] = (ll)a[i][j]*d%mod;
        _rfor(j, i+1, n){
            d = a[j][i]; a[j][i] = 0;
            if (d) _rfor(k, i+1, n+1) a[j][k] = (a[j][k]+mod-(ll)d*a[i][k]%mod)%mod
        }
    }
    _dfor(i, n-1, 2) _dfor(j, i-1, 1)
        a[j][n+1] = (a[j][n+1]+mod-(ll)a[i][n+1]*a[j][i]%mod)%mod, a[j][i] = 0;
    return true;
}
```

浮点数版

```
bool gauss(double a[maxn][maxn], int n){
    _rfor(i, 1, n){
        double d; int tar = 0;
        _rfor(j, i, n) if (fabs(a[j][i]) > dlt){ tar = j; break; }
        if (!tar) return false;
        if (tar != i) _rfor(j, i, n+1) swap(a[tar][j], a[i][j]);
        _rfor(j, i+1, n+1) a[i][j] = a[i][j]/a[i][i];
        a[i][i] = 1.0;
        _rfor(j, i+1, n){
            d = a[j][i]; a[j][i] = 0.0;
            _rfor(k, i+1, n+1) a[j][k] = a[j][k] - a[i][k]*d;
        }
    }
    _dfor(i, n-1, 2) _dfor(j, i-1, 1)
        a[j][n+1] = a[j][n+1] - a[i][n+1]*a[j][i], a[j][i] = 0.0;
    return true;
}
```

增广矩阵逆元版

```
int matrix[maxn][maxm];
bool gauss(int a[maxn][maxn], int n, int m){
    _rfor(i, 1, n){
        int d, tar = 0;
        _rfor(j, i, n) if (a[j][i] != 0){ tar = j; break; };
        if (tar != i) _rfor(j, i, m) swap(a[tar][j], a[i][j]);
        d = get_inv(a[i][i]); a[i][i] = 1;
        _rfor(j, i+1, m) a[i][j] = (ll)a[i][j]*d%mod
        _rfor(j, i+1, n){
            d = a[j][i]; a[j][i] = 0;
            if (d) _rfor(k, i+1, m) a[j][k] = (a[j][k]+mod-(ll)a[i][k]*d)%mod;
        }
    }
    _dfor(i, n-1, 2) _dfor(j, i-1, 1){
        _rfor(k, n+1, m) a[j][k] = (a[j][k]+mod-(ll)a[i][k]*a[j][i]%mod)%mod;
        a[j][i] = 0;
    }
   return true;
}
```

行列式

```
void swap(int *a, int *b, int n){
    static int tmp[maxn];
    unsigned int s = sizeof(int)*n;
    memcpy(tmp, a, s);
    memcpy(a, b, s);
    memcpy(b, tmp, s);
}

int get_det(int mtx[maxn][maxn], int n){
    int sgn = 1, tar = 0, d = 0;
    _rfor(i, 1, n){
        tar = 0;
        _rfor(j, i, n) if (a[j][i]){ tar = j; break; }
        if (!tar) return 0;
        if (tar != i) swap(a[i]+i, a[tar]+i, n-i+1), sgn = -sgn;
        _rfor(j, i+1, n){
            while(a[j][i]){
                d = a[i][i]/a[j][i]; a[i][i] %= a[j][i];
                if (d) _rfor(k, i+1, n) a[i][k] = (a[i][k]+mod-(ll)a[j][k]*d%mod)%mod;
                swap(a[i]+i, a[j]+i, n-i+1), sgn = -sgn;
            }
        }
    }
    d = sgn+mod;
    _rfor(i, 1, n) d = (ll)d*a[i][i]%mod;
    return (d+mod)%mod;
}
```

矩阵求逆

增广矩阵 (A, I_n) 经过高斯消元后可得到 (I_n, A^{-})，其中 I_n 是 n 阶单位矩阵，A 是 n 阶举证，代码实现上就是增广矩阵高版的高斯消元。

计算几何

辅助计算

```
#define eps 1e-6
#define eq(x, y)  ((y)-eps <= (x) && (x) <= (y)+eps)
#define cl(x, y) (x + eps < y)
#define cle(x, y) (x <= y + eps)
#define cr(x, y) cl(y, x)
#define cre(x, y) cr(y, x)
bool zr(db x){ return -eps <= x && x <= eps; }
int sgn(db x){ return x > eps ? 1 : x < -eps ? -1 : 0; }
db sqr(db x){ return x*x; }
```

2DVector

```
struct Point{
    db x, y;
    Point operator*(db d)const{ return {x*d, y*d}; }
    Point operator/(db d)const { return {x/d, y/d}; }
    Point operator+(const Point &t)const{ return {x+t.x, y+t.y}; }
    Point operator-(const Point &t)const{ return {x-t.x, y-t.y}; }
    Point operator*(const Point &t)const{ return {x*t.x, y*t.y}; }
    Point operator/(const Point &t)const{ return {x/t.x, y/t.y}; }
    db operator|(const Point &t)const{ return x*t.x+y*t.y; }
    db operator^(const Point &t)const{ return x*t.y-y*t.x; }
    bool operator==(const Point &t)const{ return (*this-t).dis() <= eps; }
    db dis()const{ return sqrt(x*x+y*y); }
};
db dis(const Point &a){ return sqrt(a.x*a.x+a.y*a.y); }
db dis(const Point &a, const Point &b){ return sqrt((a.x-b.x)*(a.x-b.x)+(a.y-b.y)*(a.y-b.y)); }
db cos(const Point &a, const Point &b){ return (a|b)/a.dis()/b.dis(); }
Point rotate(const Point &a, db t){ db c = cos(t), s = sin(t); return {a.x*c-a.y*s, a.x*s+a.y*c}; }
Point norm(const Point &a){ return a/a.dis(); } //a 不能为原点
bool anticlock(const Point &a, const Point &b, const Point &c){ return b-a^c-b >= -eps; }
bool clockwise(const Point &a, const Point &b, const Point &c){ return b-a^c-b <= eps; }

struct Line{
    Point a, b;
    inline db mix()const{ return min(a.x, b.x); }
    inline db miy()const{ return min(a.y, b.y); }
    inline db mxx()const{ return max(a.x, b.x); }
    inline db mxy()const{ return max(a.y, b.y); }
}; //要保证 a != b
db len(const Line &l){ return dis(l.a, l.b); }
bool onl0(const Point &p, const Line &l){ return zr((p-l.a)^(l.b-p)); }
bool onl2(const Point &p, const Line &l){
    return onl0(p, l) &&
        cle(l.mix(), p.x) && cle(p.x, l.mxx()) &&
        cle(l.miy(), p.y) && cle(p.y, l.mxy());
}
db dis0(const Point &p, const Line &l){ return fabs(((l.a-p)^(l.b-p))/dis(l.a-l.b)); }
db dis2(const Point &p, const Line &l){
    if (sgn(p-l.a | l.b-l.a) <= 0) return dis(p, l.a);
    else if (sgn(p-l.b | l.a-l.b) <= 0) return dis(p, l.b);
    else return dis0(p, l);
}
bool para(const Line &l1, const Line &l2){ return zr(l1.b-l1.a ^ l2.b-l2.a); }
bool on_right(const Point &p, const Line &l){ return clockwise(l.a, l.b, p); }
bool on_left(const Point &p, const Line &l){ return anticlock(l.a, l.b, p); }
Point inter(const Line &l1, const Line &l2){ //需要保证 l1 和 l2 不平行
    db ls = l1.b-l1.a ^ l2.a-l1.a, rs = l1.b-l1.a ^ l2.b-l1.a;
    return l2.a+(l2.b-l2.a)*(ls/(ls-rs));
}
bool is_inter(const Line &l1, const Line &l2){
    if (
        cl(l1.mxx(), l2.mix()) || cl(l1.mxy(), l2.miy()) ||
        cl(l2.mxx(), l1.mix()) || cl(l2.mxy(), l1.miy())) return false;
    else return
        sgn(l1.b-l1.a^l2.a-l1.a)*sgn(l1.b-l1.a^l2.b-l1.a) <= 0 &&
        sgn(l2.b-l2.a^l1.a-l2.a)*sgn(l2.b-l2.a^l1.b-l2.a) <= 0;
}
```

ConvexHull

```
int hn;
Point hull[maxn], tmp[2][maxn];
bool anticlock(const Point &a, const Point &b, const Point &c){ return (b-a^c-b) >= -
eps; }
bool clockwise(const Point &a, const Point &b, const Point &c){ return (b-a^c-b) <=
eps; }
bool convex_hull_cmp(const Point &p1, const Point &p2){
    return eq(p1.x, p2.x) ? cl(p1.y, p2.y) : cl(p1.x, p2.x);
}
void get_hull(Point *hull, int &hn, Point *pnt, int n){ //同时求出上凸壳和下凸壳
    sort(pnt+1, pnt+1+n, convex_hull_cmp);
    db Y[2] = {min(pnt[1].y, pnt[n].y), max(pnt[1].y, pnt[n].y)};
    int t[2] = {0, 0}, s[2] = {-1, 1};
    _rfor(i, 1, n){
        Point &p = pnt[i];
        _for(o, 0, 2){
            if (!o && cl(p.y, Y[o]) || o && cr(p.y, Y[o])) continue;
            while(t[o] >= 2 && sgn(tmp[o][t[o]]-tmp[o][t[o]-1]^p-tmp[o][t[o]]) != s[o])
--t[o];
            tmp[o][++t[o]] = p;
        }
    }
    --t[1];
    hn = t[0] + t[1];
    _rfor(i, 1, t[0]) hull[i] = tmp[0][i];
    _rfor(i, 1, t[1]) hull[i+t[0]] = tmp[1][t[1]-i+1];
}

void get_hull(Point *hull, int &hn, Point *pnt, int n){//直接求出整个凸包
    sort(pnt+1, pnt+1+n, convex_hull_cmp);
    db Y1 = min(pnt[1].y, pnt[n].y), Y2 = max(pnt[1].y, pnt[n].y);
    int k = 2; hn = 0;
    _for(o, 0, 2){
        _rfor(i, 1, n-o){
            Point &p = pnt[o?n-i:i];
            if (!o && cl(p.y, Y1) || o && cr(p.y, Y2)) continue;
            while(hn >= k && anticlock(hull[hn-1], hull[hn], p)) --hn;
            hull[++hn] = p;
        }
        if (!o) mid = hn;
        k = hn+1;
    }
}
```

MinkowskiSum

```
void get_sum_hull(Point h[3][maxn], int hn[3]){
    static Point hull[maxn];
    int i[2] = {2, 2}, t = 0;
    hull[++t] = h[0][1] + h[1][1];
    while(i[0] <= hn[0] || i[1] <= hn[1]){
        int o = i[0] > hn[0] || i[1] <= hn[1] &&
            sgn(h[0][i[0]]-h[0][i[0]-1]^h[1][i[1]]-h[1][i[1]-1]) > 0;
        hull[t+1] = h[o][i[o]]-h[o][i[o]-1]+hull[t];
        ++t; ++i[o];
    }
    get_hull(h[2], hn[2], hull, t); //再求一次凸包，去掉平行边
}
```

旋转卡壳

以下是求最远点之间距离的代码段。

```
get_hull(hull, hn, pnt, n);

int i = 1, j = mid;
int ans = dis(hull[i]-hull[j]);

while(i < mid || j < hn){
    if (j == hn || i < mid &&
        (hull[i+1]-hull[i]^hull[j]-hull[j+1]) <= eps) ++i;
    else ++j;
    ans = max(ans, dis(hull[i]-hull[j]));
}
```

半平面交

```
struct Line{
    Point a, b; db theta;
    void get_theta(){ theta = atan2((b-a).y, (b-a).x); }
} lne[maxn];

bool clockwise(const Point &a, const Point &b, const Point &c);
bool on_right(const Point &p, const Line &l);
bool para(const Line &l1, const Line &l2);
Point inter(const Line &l1, const Line &l2);

bool lne_cmp(const Line &l1, const Line &l2){
    if (eq(l1.theta, l2.theta)) return !right(l1.a, l2);
    else return l1.theta < l2.theta;
}

Line que[maxn]; Point tmp[maxn];
int semi_l, semi_r; bool semi_zero = false;
void semi_plane(Line *lne, int n){
    int &l = semi_l, &r = semi_r;
    _rfor(i, 1, n) lne[i].get_theta();
    sort(lne+1, lne+1+n, lne_cmp);

    que[r] = lne[1];
    _rfor(i, 2, n){
        if (eq(lne[i].theta, lne[i-1].theta)) continue;
        Line &ln = lne[i];
        while(r > l && right(tmp[r], ln)) --r;
        while(r > l && right(tmp[l+1], ln)) ++l;
        if (para(ln, que[r])){ semi_zero = true; return; }
        que[++r] = ln;
        tmp[r] = inter(que[r-1], ln);
    }
    while(r > l && right(tmp[r], que[l])) --r;
    tmp[l] = inter(que[r], que[l]);
}
```

斜率版半平面交

```
#define KEPS 1e-6
#define ANS_INIT 1e18
struct Line{
    double k, b, t;
    Line():t(-ANS_INIT){}
    double cal(double x){return k*x+b;}
    double inter(const Line &t)const{
        assert(fabs(k-t.k) > KEPS);
        return (t.b-b)/(k-t.k);
    }
};

bool nz(double x){ return x<0.0?x<-KEPS:x>KEPS;}

struct Stk{
    int sg;
    Line l[maxn], q[maxn];
    int lcnt, qcnt;
    Stk():lcnt(0),sg(-1){}
    void init(int sg){
        qcnt = lcnt = 0; this->sg = sg;
    }
    void add(const Line&t){
        l[lcnt++] = t;
    }
    void build(){
        if (sg == -1){
            sort(l, l+lcnt, [](const Line&l1, const Line&l2){
                return (nz(l1.k-l2.k)?l1.k>l2.k:l1.b<l2.b);
            });
        }
        else{
            sort(l, l+lcnt, [](const Line&l1, const Line&l2){
                return (nz(l1.k-l2.k)?l1.k<l2.k:l1.b>l2.b);
            });
        }

        _for(i, 0, lcnt){
            Line ln = l[i];
            if (i > 0 && !nz(l[i].k-l[i-1].k)) continue;
            while(qcnt){
                Line lln = q[qcnt-1];
                assert(nz(lln.k-ln.k));
                double x = lln.inter(ln);
                if (x-KEPS <= lln.t) --qcnt;
                else break;
            }
            if (qcnt == 0){
                ln.t = ninf;
                q[qcnt++] = ln;
            }
            else{
                Line lln = q[qcnt-1];
                ln.t = lln.inter(ln);
                assert(ln.t-KEPS > lln.t);
                q[qcnt++] = ln;
            }
        }
    }
    double cal(double x){
        int as = qcnt, le = 0, ri = qcnt, mi;
        //pf("qcnt:%d\n", qcnt);

        while(le < ri){
            mi = le+ri>>1;
            if (x+KEPS > q[mi].t) as = mi, le = mi+1;
            else ri = mi;
        }
        assert(as < qcnt);
        return q[as].cal(x);
    }
};
```