SCL

Cothrax

October 8, 2019

Contents

1 Emacs Profile

```
(show-paren-mode t)
(global-linum-mode 1)
(setq default-tab-width 4)
(setq c-basic-offset 4)
(set-face-attribute 'default nil :height 120)

;; DOSKEY g++=g++ -Wall -Wextra -g -DLOCAL -Wshadow -Wpointer-arith -Wcast-qual
↪  -Winline -Wunreachable-code
;; alias g++=g++ -Wall -Wextra -g -DLOCAL -Wshadow -Wpointer-arith -Wcast-qual
↪  -Winline -Wunreachable-code
```

2 ./1-head-short.cpp

```
#include<bits/stdc++.h>
#define rep(i, l, r) for(ll i = (l); i <= (r); i++)
#define per(i, r, l) for(ll i = (r); i >= (l); i--)
#define foreach(i, x) for(auto i = x.begin(); i != x.end(); i++)
#define forG(i) for(int i = head[u], v; i; i = g[i].nxt)
typedef long long ll;
using namespace std;
#define X first
#define Y second
#define mkp(x, y) make_pair(x, y)
#define pb(x) push_back(x)
#define INF (ll)0x3f3f3f3f
#define EPS 1e-8
template<class T> inline void read(T &ret)
{
    T x = 0, f = 1; char ch = getchar();
    while(ch < '0' || ch > '9') {if(ch == '-') f = -1; ch = getchar();}
    while(ch >= '0' && ch <= '9'){x = x*10+ch-'0'; ch = getchar();}
    ret = x*f;
}

template<class T> inline T lowbit(T x) {return x&(-x);}
template<class T> T gcd(T a, T b) {return b?gcd(b,a%b):a;}
template<class T> inline T Pow(T a, T b, T p)
{T ret=1;a%=p;for(;b;b>>=1,a=a*a%p)if(b&1)(ret*=a)%=p;return ret;}
#define disp0(A){foreach(i,A)cout<<A[i]<<" ";cout<<endl;}
#define disp(A, l, r) {rep(_i,l,r)cout<<A[_i]<<" ";cout<<endl;}
#define disp2(A, l, r, b, e){                                          \
        rep(_i,l,r){rep(_j,b,e)cout<<A[_i][_j]<<"\t";cout<<endl;}     \
        cout<<endl;                                                    \
    }
```

3 ./2-data-structure/block.cpp

```
/*
    分块 (例为静态区间众数)
 */

#define I(x) min((x)*blo, n)
const int N = 10010, M = 410;
int n, m, a[N]; vector<int> seq; P b[N];
void disc()
{
    sort(seq.begin(), seq.end());
    seq.erase(unique(seq.begin(), seq.end()), seq.end());
    rep(i, 0, n-1) a[i] = lbound(seq.begin(), seq.end(), a[i])-seq.begin();
}

int T = 0, flg[N], cnt[N], max_c, cur;
void upd(int x)
{
    if(flg[x] != T) cnt[x] = 0, flg[x] = T;
    cnt[x]++;
    if(cnt[x]>max_c || (cnt[x]==max_c && x<cur))
        cur = x, max_c = cnt[x];
}

int bl[N], blo, sz, f[M][M], g[M][M];
void init()
{
    rep(i, 0, n-1) b[i] = mkp(a[i], i);
    sort(b, b+n);

    blo = ceil(sqrt(n));
    rep(i, 0, n-1) bl[i] = i/blo;
    rep(i, 0, bl[n-1])
    {
        T++; max_c = 0; cur = 0;
        rep(j, I(i), n-1)
        {
            upd(a[j]);
            if(j==n-1 || (j+1)%blo == 0)
                f[i][bl[j]] = cur, g[i][bl[j]] = max_c;
        }
    }
}

int stk[N], top = 0; bool vis[N];
int calc(int l, int r)
{
    rep(i, l, min(I(bl[l]+1)-1, r))
        if(!vis[a[i]]) vis[a[i]] = 1, stk[++top] = a[i];
    if(bl[l] != bl[r])
        rep(i, I(bl[r]), r)
```

```
            if(!vis[a[i]]) vis[a[i]] = 1, stk[++top] = a[i];
    int ret = f[bl[l]+1][bl[r]-1], cur_c = g[bl[l]+1][bl[r]-1];
    rep(i, 1, top)
    {
        int lb = lbound(b, b+n, mkp(a[i], l)) - b,
            ub = ubound(b, b+n, mkp(a[i], r)) - b,
            len = ub-lb;
        if(len>cur_c || (len==cur_c && a[i]<ret)) cur_c = len, ret = a[i];
    }
    return ret;
}

int main()
{
    freopen("std.in", "r", stdin);
    // freopen("std.out", "w", stdout);
    read(n); read(m);
    rep(i, 0, n-1) read(a[i]), seq.pb(a[i]);
    disc(); init();

    // disp2(f, 0, blo, 0, blo);
    // disp2(g, 0, blo, 0, blo);

    int ans = 0;
    rep(i, 1, m)
    {
        int l, r; read(l); read(r);
        // l = (l+ans-1)%n; r = (r+ans-1)%n; if(l>r) swap(l, r);
        l--; r--;
        printf("%d\n", ans = seq[calc(l, r)]);
    }
}
```

4 ./2-data-structure/dsu.cpp

```
struct dsu
{
    int par[N], siz[N]; //siz[i] 为 i 所在集合的大小
    void init(int n)
        {
            for(int i = 1; i <= n; i++) par[i] = i, siz[i] = 1;
        }
    int find(int u) { return par[u]==u ? u : par[u] = find(par[u]); }
    void join(int u, int v) //u -> v
        {
            u = find(u); v = find(v);
            if(u == v) return;
            par[u] = v; siz[v] += siz[u];
        }
    int getsz(int u) {return siz[find(u)];}
} s;
```

5 ./2-data-structure/odt.cpp

```
struct ODT
{
    struct node
    {
        int l, r; mutable ll v;
        node(int _l, int _r, ll _v): l(_l), r(_r), v(_v) {}
        inline bool operator<(const node &o) const { return l < o.l; }
    };
    set<node> odt;
    int n;
    void init(int _n, ll *w)
    {
        n = _n;
        odt.clear();

        int lst = 1;
        rep(i, 2, n)
            if(w[i] != w[i-1])
            {
                odt.insert(node(lst, i-1, w[i-1]));
                lst = i;
            }
        odt.insert(node(lst, n, w[n]));
    }

    auto split(int x)
    {
        if(x > n) return odt.end();
        auto it = --odt.upper_bound(node(x, 0, 0));
        if(it->l == x) return it;
        int l = it->l, r = it->r; ll v = it->v;
        odt.erase(it);
        odt.insert(node(l, x-1, v));
        return odt.insert(node(x, r, v)).X;
    }
    void assign(int l, int r, ll v)
    {
        auto itr = split(r+1), itl = split(l);
        odt.erase(itl, itr);
        odt.insert(node(l, r, v));
    }
    void inc(int l, int r, ll k)
    {
        auto itr = split(r+1), itl = split(l);
        for(; itl != itr; ++itl) itl->v += k;
    }
    ll kth(int l, int r, int k)
    {
        vector<pair<ll, int> > v;
        auto itr = split(r+1), itl = split(l);
        for(; itl != itr; ++itl) v.push_back(mkp(itl->v, itl->r-itl->l+1));
        sort(v.begin(), v.end());
        int pref = 0;
        foreach(j, v)
        {
            pref += j->Y;
            if(pref >= k) return j->X;
        }
    }
} t;
```

6 ./2-data-structure/offline/cdq.cpp

```
/*
  三维偏序 - cdq 分治 O(N(logN)^2)
  例为 luogu p3157 动态逆序对
  note: 第一维 t 排序
        第二维 pos 归并
        第三维 val 用两个 bit 维护
  func: cdq(l, r)
 */

const int N = 100010;
struct node { int t, pos, val; } a[N], tmp[N];
bool comp(node x, node y) { return x.t < y.t; }
int n, m; ll ans[N];

struct BIT
{
    ll bit[N];
    void add(int x, ll k)
        {
            if(!x) return;
            for(; x < N; x += lowbit(x)) bit[x] += k;
        }
    ll ask(int x)
        {
            ll ret = 0;
            for(; x; x -= lowbit(x)) ret += bit[x];
            return ret;
        }
} low, high;

void cdq(int l, int r)
{
    if(l == r) return;
    int mid = (l+r)>>1, p = l, q = mid+1;
    cdq(l, mid); cdq(mid+1, r);
    rep(i, mid+1, r) low.add(a[i].val, 1);
    rep(i, l, r)
        if(p>mid || (q<=r && a[q].pos<a[p].pos))
        {
            tmp[i] = a[q];
            low.add(a[q].val, -1); high.add(a[q++].val, 1);
        }
        else
        {
            tmp[i] = a[p];
            ll cur = low.ask(a[p].val)+high.ask(n+1)-high.ask(a[p].val);
            // printf("[%d, %d] p=%d, q=%d cur=%d\n", l, r, p, q, cur);
            ans[0] += cur; ans[a[p++].t] -= cur;
        }
    rep(i, mid+1,r) high.add(a[i].val, -1);
    rep(i, l, r) a[i] = tmp[i];
}

int main()
{
    freopen("std.in", "r", stdin);
    freopen("std.out", "w", stdout);
    read(n); read(m);
    rep(i, 1, n) a[i].val = i;
    rep(i, 1, n) { int x; read(x); a[x].pos = i; }
    rep(i, 1, m) { int x; read(x); a[x].t = i; }
    rep(i, 1, n) if(!a[i].t) a[i].t = n+1;
    sort(a+1, a+n+1, comp);
    cdq(1, n);
    rep(i, 1, m) ans[i] += ans[i-1];
    rep(i, 0, m-1) printf("%lld\n", ans[i]);
}
```

7 ./2-data-structure/offline/mo.cpp

```
/*
  普通莫队 O(N^(3/2)) (例为离线区间众数)
*/
namespace moq1
{
    int n, m, lst, b[N], bl[N], blo; P a[N];
    struct node { int x, y, i; } ask[N];
    bool comp(node a, node b)
    { return (bl[a.l]<bl[b.l])||(bl[a.l]==bl[b.l]&&a.r<b.r); }

    int ans[N], num[N], cnt[N], cur = 0;
    void inc(int x)
    {
        if(cur == num[x]) cur++;
        cnt[num[x]++]--; cnt[num[x]]++;
    }
    void dec(int x)
    {
        if(cur == num[x] && cnt[num[x]] == 1) cur--;
        cnt[num[x]--]--; cnt[num[x]]++;
    }

    void solv()
    {
        n = read(); m = read(); blo = sqrt(n)+1;
        for(int i = 1; i <= n; i++) bl[i] = (i-1)/blo; //分块编号
        for(int i = 1; i <= n; i++) fi(a[i]) = read(), se(a[i]) = i;
        for(int i = 1; i <= m; i++)
            ask[i].l = read(), ask[i].r = read(), ask[i].i = i;
        sort(ask + 1, ask + m + 1, comp);

        sort(a + 1, a + n + 1); lst = 0; //离散化
        for(int i = 1; i <= n; i++)
        {
            if(i == 1 || fi(a[i]) != fi(a[i-1])) lst++;
            b[se(a[i])] = lst;
        }

        int l = 1, r = 0; //two-pointer
        for(int i = 1; i <= m; i++)
        {
            while(l < ask[i].l) dec(b[l++]); //注意映射关系！
            while(r < ask[i].r) inc(b[++r]);
            while(l > ask[i].l) inc(b[--l]);
            while(r > ask[i].r) dec(b[r--]);
            ans[ask[i].i] = cur; //注意 ask[i].i!
        }
        for(int i = 1; i <= m; i++) printf("%d\n", ans[i]);
    }
}

/*
  带修莫队 (例为统计区间不同元素个数) O(N^(5/3))
  var:  bl[N], blo            : 分块编号, 块数
        cnt[N], tot           : 计数变量
        p, q                  : 查询数, 修改数
        col[N]                : 原序列
        idx[N], val[N]        : 修改信息, 第 x 个为将 idx[x] 改为 val[x]
        ask[N]{l,r,tim,i}     : 查询信息
  func: inc(c)                : 增点
        dec(c)                : 删点
        modify(x, l, r)       : 执行第 x 次修改, 当前区间为 [l, r]
        unmod(x, l, r)        : 撤销第 x 次修改, 当前区间为 [l, r]
        solv()                : 接口
 */
namespace moq2
{
    const int N = 2e6+10;
    struct node {int l, r, tim, i; } ask[N];
```

```
int bl[N], blo;
bool comp(node a, node b)
{
    return (bl[a.l]==bl[b.l]) ?
        (bl[a.r]==bl[b.r] ? a.tim<b.tim : a.r<b.r) : a.l<b.l;
}

int cnt[N], tot = 0;
int lstv[N], idx[N], val[N], col[N], pre[N];
void inc(int c) { cnt[c]++; if(cnt[c] == 1) tot++; }
void dec(int c) { cnt[c]--; if(!cnt[c]) tot--; }
inline void modify(int x, int l, int r)
{
    col[idx[x]] = val[x];
    if(idx[x]<l || r<idx[x]) return; //修改点不在当前区间中
    dec(lstv[x]); inc(val[x]);
}
inline void unmod(int x, int l, int r)
{
    col[idx[x]] = lstv[x];
    if(idx[x]<l || r<idx[x]) return;
    dec(val[x]); inc(lstv[x]);
}

int p, q, n, m, ans[N];
void solv()
{
    freopen("std.in", "r", stdin);
    // freopen("std.out", "w", stdout);
    read(n); read(m); blo = pow(n, 2.0/3);
    rep(i, 1, n) read(col[i]);
    rep(i, 1, n) bl[i] = i/blo;
    p = q = 0;
    rep(i, 1, m)
    {
        char c; scanf("%c", &c);
        if(c == 'Q')
        {
            read(ask[++p].l); read(ask[p].r);
            ask[p].i = p; ask[p].tim = q;
        }
        else
        {
            read(idx[++q]); read(val[q]); int x = idx[q];
            lstv[q] = pre[x] ? val[pre[x]] : col[x]; pre[x] = q;
        }
    }
    sort(ask+1, ask+p+1, comp);
    int l = 1, r = 0, cur = 0;
    rep(i, 1, p)
    {
        while(cur < ask[i].tim) modify(++cur, l, r);
```

```
            while(cur > ask[i].tim) unmod(cur--, l, r);
            while(l > ask[i].l) inc(col[--l]);
            while(r < ask[i].r) inc(col[++r]);
            while(l < ask[i].l) dec(col[l++]);
            while(r > ask[i].r) dec(col[r--]);
            ans[ask[i].i] = tot;
        }
        rep(i, 1, p) printf("%d\n", ans[i]);
    }
}

/*
  处理路径询问的树上莫队 (例为询问路径上不同元素个数)
  note: 基本思路:   找出 u-v (lft[u] < lft[v]) 对应的询问区间 [l, r]
                    使 u-v 上的点都出现 1 次，其余点出现 0 次或 2 次
                    case1: 当 lca(u, v) == u 时，[lft[u], lft[v]]
                    case2: 当 lca(u, v) != u 时，[rgt[u], lft[v]] 外加 lca(u, v)
        (注意序列长度为 2*n)
  var:  N                        : 点数 *2
        M                        : 询问数
        lft[N], rgt[N], seq[N]   : dfs 进出时间戳，得到的序列 (长度 2*n)
        lca[N]                   : 第 i 个询问在 case2 下的 lca
  func: inc(c)                   : 增点
        dec(c)                   : 删点
  bouns: 处理子树询问? 普通 dfs 序即可 [dfn[u], dfn[u]+siz[u]-1]
*/
namespace moq3
{
    const int N = 40010*2, M = 100010;
    // 图部分
    struct edge{int v, nxt;} g[N];
    int head[N], sz = 0;
    void add(int u, int v)
    {
        g[++sz].v = v; g[sz].nxt = head[u]; head[u] = sz;
    }

    int n, m, a[N], bl[N], blo, lgn; int lca[M], ans[M];
    int lft[N], rgt[N], seq[N], dep[N], par[N][20], tim = 0;
    void dfs(int u, int p)
    {
        lft[u] = ++tim; seq[tim] = u;
        par[u][0] = p; dep[u] = dep[p] + 1;
        for(int i = head[u], v; i; i = g[i].nxt)
            if((v = g[i].v) != p) dfs(v, u);
        rgt[u] = ++tim; seq[tim] = u;
    }
    int lca(int u, int v)
    {
        if(dep[u] < dep[v]) swap(u, v);
        for(int i = lgn; i >= 0; i--)
            if(dep[par[u][i]] >= dep[v]) u = par[u][i];
        if(u == v) return u;
        for(int i = lgn; i >= 0; i--)
            if(par[u][i] != par[v][i])
                u = par[u][i], v = par[v][i];
        return par[u][0];
    }

    // 莫队部分
    struct node { int l, r, i; } ask[M];
    bool comp(node a, node b)
    { return (bl[a.l] < bl[b.l]) || (bl[a.l] == bl[b.l] && a.r <= b.r); }

    int cnt[N], cur = 0, flg[N];
    void _inc(int x) { if(cnt[x]++ == 0) cur++; }
    void _dec(int x) { if(--cnt[x] == 0) cur--; }
    void inc(int k) { (flg[k]++ == 1) ?  _dec(a[k]) : _inc(a[k]); }
    void dec(int k) { (flg[k]-- == 1) ?  _dec(a[k]) : _inc(a[k]); }

    void solv()
    {
        read(n); read(m); lgn = log2(n)+1; blo = sqrt(n*2);
        for(int i = 1; i <= n*2; i++) bl[i] = i/blo;
        for(int i = 1; i <= n; i++) read(a[i]);
        for(int i = 1; i < n; i++)
        {
            int u = read(), v = read();
            add(u, v); add(v, u);
        }
        init(); dfs(1, 0);
        for(int i = 1; i <= lgn; i++)
            for(int j = 1; j <= n; j++)
                par[j][i] = par[par[j][i-1]][i-1];

        for(int i = 1; i <= m; i++)
        {
            int u = read(), v = read(), p = lca(u, v);
            ask[i].i = i; if(lft[u] > lft[v]) swap(u, v);
            if(u == v) ask[i].l = ask[i].r = lft[u];
            else if(u == p) ask[i].l = lft[u], ask[i].r = lft[v];
            else ask[i].l = rgt[u], ask[i].r = lft[v], lca[i] = p;
        }
        sort(ask + 1, ask + m + 1, comp);
        int l = 1, r = 0;
        for(int i = 1; i <= m; i++)
        {
            while(l > ask[i].l) inc(seq[--l]);
            while(r < ask[i].r) inc(seq[++r]);
            while(l < ask[i].l) dec(seq[l++]);
            while(r > ask[i].r) dec(seq[r--]);
            int j = ask[i].i;
            if(lca[j]) inc(lca[j]); //单独处理 lca
            ans[j] = cur;
            if(lca[j]) dec(lca[j]);
        }
```

```
        for(int i = 1; i <= m; i++) printf("%d\n", ans[i]);
    }
}
```

8 ./2-data-structure/queue.cpp

```
/*
  单调队列 (例为定窗长最大值)
  var:  h, t          : [h, t)
        q[N]          : 元素值队列
        d[N]          : 标号队列
  func: init()        : init
        upd_h(x)      : 用左端点 x 更新队头
        upd_t(x)      : 用新元素值 x 更新队尾
        ins(x, i)     : 向队尾插入一个值为 x, 标号 i 的元素
        top()         : 返回队尾值
  note: 一般流程:    k<j:
                     1. upd_h
                     2. f[i][j] = ...q.top()...
                     3. upd_t
                     4. ins
        k<=j: 将 2 换到最后
        注意 dp 初始条件, 及 init
        empty 时务必特判
 */

struct Queue
{
    int h, t, q[N], d[N]; //d 记录从头出队信息
    void init() { h = t = 0; }
    void upd_h(int x) { while(h != t && d[h] < x) h++; }
    void upd_t(int x) { while(h != t && q[t-1] <= x)) t--; }
    void ins(int x, int i) { q[t] = x; d[t++] = i; }
    int top() { return (h != t) ? q[h] : -INF; }
} q;
```

9 ./2-data-structure/st.cpp

```
/*:
  st 表 rmq
  var:  st[i][j]          : [i, i+2^j-1] rmq
  func: init()            : 预处理 O(NlogN)
        query(l, r)       : 查询 rmq [l, r]
  note: 下标 [0, n-1] 亦可
 */
const int N = 1e6, lgN = 20;
struct ST
{
    int st[N][lgN];
    void init(int *w, int n)
```

```
        {
            int lgn = ceil(log2(n));
            rep(i, 1, n) st[i][0] = w[i];
            rep(j, 1, lgn) rep(i, 1, n)
            {
                st[i][j] = st[i][j-1]; int k = i+(1<<(j-1));
                if(k < n) st[i][j] = min(st[i][j], st[k][j-1]);
            }
        }
    ll query(int l, int r)
        {
            if(l > r) return INF;
            int len = r-l+1, t = lg2[len];
            return min(st[l][t], st[r-(1<<t)+1][t]);
        }
} rmq;
```

10 ./2-data-structure/tree/bit.cpp

```
/*
  树状数组 (例为前缀和)
  note: 常数优化可以考虑把 <N 换成 <=n
 */

struct BIT
{
    ll bit[N];
    void add(int x, ll k)
        {
            if(!x) return;
            for(; x < N; x += lowbit(x)) bit[x] += k;
        }
    ll ask(int x)
        {
            ll ret = 0;
            for(; x; x -= lowbit(x)) ret += bit[x];
            return ret;
        }
} t;
```

11 ./2-data-structure/tree/centroid-decomposition.cpp

```
/*
  点分治 (统计长度 =k 的路径, 共 m 次询问)
  var:  siz[]             : 子树大小
        q[cnt]            : 每次分治的 dist{u, rt}
        used[u]           : u 是否被 dfs 计算过
  func: get_rt(u, p, min_sz, rt, sum)    : 计算重心 (u, p, &, &, 树的总大小)
        get_dist(u, p, pre)              : 计算 dist{u, rt}
        calc(u, ext, flg)                : 计算贡献
        dfs(u)                          : 点分治
  note: 1. 每次分治时重心也要计入答案
        2. 但计入答案时要使用 ext
 */
int siz[N], q[N], cnt = 0; bool used[N];
int n, m, a[N], f[N];
void get_rt(int u, int p, int &min_sz, int &rt, int sum)
{
    int cur = 0;
    for(int i = head[u], v; i; i = g[i].nxt)
        if(!used[v = g[i].v] && v != p)
        {
            get_rt(v, u, min_sz, rt, sum);
            cur = max(cur, siz[v]);
        }
    cur = max(cur, sum - siz[u]);
    if(cur < min_sz) rt = u, min_sz = cur;
}

void get_sz(int u, int p)
{
    // printf("get_sz: %d\n", u);
    siz[u] = 1;
    for(int i = head[u], v; i; i = g[i].nxt)
        if(!used[v = g[i].v] && v != p)
        {
            get_sz(v, u);
            siz[u] += siz[v];
        }
}
void get_dist(int u, int p, int pre)
{
    for(int i = head[u], v; i; i = g[i].nxt)
        if((v = g[i].v) != p && !used[v])
        {
            q[++cnt] = pre + g[i].w;
            get_dist(v, u, q[cnt]);
        }
}
void calc(int u, int ext, int flg)
{
    q[cnt = 1] = ext; get_dist(u, 0, ext);
    sort(q + 1, q + cnt + 1);
    for(int i = 1; i <= m; i++) //原题要统计 m 个 k
    {
        int ptr = 1;
        while(ptr < cnt && q[ptr] + q[cnt] < a[i]) ptr++;
        while(ptr < cnt && q[ptr] * 2 <= a[i])
        {
            int lb = lower_bound(q+ptr+1, q+cnt+1, a[i]-q[ptr]) - q,
                ub = upper_bound(q+ptr+1, q+cnt+1, a[i]-q[ptr]) - q;
            if(lb < ub) f[i] += (ub-lb)*flg;
            ptr++;
        }
    }
}
void dfs(int u)
{
    used[u] = 1; calc(u, 0, 1);
    get_sz(u, 0);
    for(int i = head[u], v; i; i = g[i].nxt)
        if(!used[v = g[i].v])
        {
            calc(v, g[i].w, -1);
            int min_sz = n+1, rt = 0;
            get_rt(v, u, min_sz, rt, siz[v]); dfs(rt);
        }
}

int main()
{
    int min_sz = n+1, rt = 0;
    get_sz(1, 0);
    get_rt(1, 0, min_sz, rt, n);
    dfs(rt);
}
```

12 ./2-data-structure/tree/fortile.cpp

```
/*
  主席树 (例为维护区间权值和)
  func: insert(&i, b, e, k, x)   : 单点修改
        query(i, b, e, k)        : 查询 <=k 的权值和
 */

struct Tree
{
    struct node { int l, r, w; } sgt[N*20];
    int p = 0, rt[N] = {0};
    void insert(int &i, int b, int e, int k, int x)
        {
            sgt[++p] = sgt[i]; i = p;
            sgt[i].w = max(sgt[i].w, x);
            if(b==e) return;
            int mid = (b+e)>>1;
            if(k<=mid) insert(sgt[i].l, b, mid, k, x);
            else insert(sgt[i].r, mid+1, e, k, x);
        }
    int query(int i, int b, int e, int k)
        {
            if(!i) return 0;
            if(b==e) return sgt[i].w;
            int mid = (b+e)>>1;
            if(k<=mid) return query(sgt[i].l, b, mid, k);
            else return max(sgt[sgt[i].l].w, query(sgt[i].r, mid+1, e, k));
        }
} t;
```

13 ./2-data-structure/tree/hld.cpp

```
/*
  树链剖分 (例为维护路径点权和, 及最大值)
  var:  son[]                        : 重儿子
        top[]                        : 所在树链顶端节点
        dfn[], idx[]                 : dfs 序, idx[dfn[u]] = u
  func: dfs(u, p), dfs2(u, top)      : 树剖核心过程
        query(u, v, flg)             : flg==0 求 max, flg==1 求 sum
        lca(u, v)                    : 求 lca
  bonus: 维护边权? w[u] = w(par[u]->u), query 时排除 lca
        维护子树? 树剖本身也是 dfs 序
        query 有顺序要求? 先求出 lca(u, v), 分开做, 注意 lca 只能算一次
 */

int par[N], dep[N], siz[N], son[N], top[N], idx[N], dfn[N], w[N], tot=0;
void dfs(int u, int p)
{
    par[u] = p; dep[u] = dep[p] + 1; siz[u] = 1;
    int cur = 0, v;
    for(int i = head[u], v; i; i = g[i].nxt)
        if((v = g[i].v) != p)
        {
            dfs(v, u); siz[u] += siz[v];
            if(siz[v] > cur) cur = siz[v], son[u] = v;
        }
}
void dfs2(int u, int tp)
{
    dfn[u] = ++tot; idx[tot] = u; top[u] = tp;
    if(son[u]) dfs2(son[u], top[u]);
    for(int i = head[u], v; i; i = g[i].nxt)
        if((v = g[i].v) != par[u] && v != son[u]) dfs2(v, v);
}

//线段树
struct Tree
{
    int sum[N*4], max_w[N*4];
    void build(int i, int b, int e)
        {
            if(b == e) { sum[i] = max_w[i] = w[idx[b]]; return; }
            int mid = (b+e)>>1, lc = i<<1, rc = lc|1;
            build(lc, b, mid); build(rc, mid+1, e);
            sum[i] = sum[lc] + sum[rc];
            max_w[i] = max(max_w[lc], max_w[rc]);
        }
    int query_sum(int i, int b, int e, int l, int r)
        {
            if(e<l || r<b) return 0;
            if(l<=b && e<=r) return sum[i];
            int mid = (b+e)>>1, lc = i<<1, rc = lc|1;
            return query_sum(lc, b, mid, l, r) +
                query_sum(rc, mid+1, e, l, r);
        }
    int query_max(int i, int b, int e, int l, int r)
        {
            if(r<l || e<l || r<b) return -INF;
            if(l<=b && e<=r) return max_w[i];
            int mid = (b+e)>>1, lc = i<<1, rc = lc|1;
            return max(query_max(lc, b, mid, l, r),
                       query_max(rc, mid+1, e, l, r));
        }
    void upd(int i, int b, int e, int x, int k)
        {
            if(b == e) { sum[i] = max_w[i] = k; return; }
            int mid = (b+e)>>1, lc = i<<1, rc = lc|1;
            if(x<=mid) upd(lc, b, mid, x, k);
            else upd(rc, mid+1, e, x, k);
            sum[i] = sum[lc]+sum[rc];
            max_w[i] = max(max_w[lc], max_w[rc]);
        }
    void walk(int i, int b, int e, int dep)
        {
            int mid = (b+e)>>1, lc = i<<1, rc = lc|1;
            if(b!=e) walk(lc, b, mid, dep+1);
            for(int i = 0; i < dep; i++) printf("  ");
            printf("%d[%d,%d]%d;%d\n", i, b, e, sum[i], max_w[i]);
            if(b!=e) walk(rc, mid+1, e, dep+1);
        }
} t;

//查询, flg==1 求 sum, flg==0 求 max
int query(int u, int v, int flg)
{
    int p = top[u], q = top[v], ret = flg ? 0 : -INF;
    while(p != q)
    {
        if(dep[p] < dep[q]) swap(p, q), swap(u, v);
        if(flg) ret += t.query_sum(1, 1, tot, dfn[p], dfn[u]);
        else ret = max(ret, t.query_max(1, 1, tot, dfn[p], dfn[u]));
        u = par[p]; p = top[u];
    }
    if(dep[u] > dep[v]) swap(u, v);
    if(flg) ret += t.query_sum(1, 1, tot, dfn[u], dfn[v]);
    else ret = max(ret, t.query_max(1, 1, tot, dfn[u], dfn[v]));
    return ret;
}

int lca(int u, int v)
```

```
{
    int p = top[u], q = top[v];
    while(p != q)
    {
        if(dep[p]<dep[q]) swap(p, q), swap(u, v);
        u = par[p]; p = top[u];
    }
    return dep[u]<dep[v] ? u : v;
}

//若 u->v 有顺序要求则先求 lca, 在将 u->lca 和 lca->v 分开做
int calc(int u, int v, int x)
{
    int p = lca(u, v);
    //more here
    //注意 lca 不可以处理两次, 在 lca->v 中 dfn[lca]+1
}

int main()
{
    dfs(1, 0); dfs2(1, 1); //注意是 1
    t.build(1, 1, tot);
}
```

14 ./2-data-structure/tree-in-tree/bit-seg.cpp

```
/*
  带内存回收的带修主席树
  var:  MAX                 最大允许内存
        mem[MAX]            管理内存的栈
        tot                 mem 栈顶指针
  func: init()              初始化 mem
        push(x)             回收一个节点
        pop()               return 一个新节点

        ins(&i, b, e, x, k  权值线段树 - x 处单点插值
        query(i, b, e, l, r)         - [l, r] 区间查询
        ask(a, b, c, d)     二维数点 bit 中 [a, b] 线段树中 [c, d]
        modify(x, val, k)   单点修改, (x, val) 位置 +k
*/

const int N = 2e5+10, lgN = ceil(log2(N));
const int MAX = 32250007; // 手动测试最大值
int n, m;
struct Tree
{
    int rt[N];
    struct node { int l, r, w; } sgt[MAX];

    int mem[MAX], tot = 0;
    void init() { rep(i, 0, MAX-1) mem[i] = i; tot = MAX; }
    inline void push(int x) { mem[tot++] = x; }
    inline int pop() { return mem[--tot]; }

    void ins(int &i, int b, int e, int x, int k)
        {
            if(!i) i = pop(); sgt[i].w += k;
            if(b < e)
            {
                int mid = (b+e)>>1;
                if(x <= mid) ins(sgt[i].l, b, mid, x, k);
                else ins(sgt[i].r, mid+1, e, x, k);
            }
            if(!sgt[i].w)  { sgt[i] = {0, 0, 0}; push(i); i = 0; }
        }
    int query(int &i, int b, int e, int l, int r)
        {
            if(!i || e < l || r < b) return 0;
            if(l <= b && e <= r) return sgt[i].w;
            int mid = (b+e)>>1, lc = sgt[i].l, rc = sgt[i].r;
            return query(lc, b, mid, l, r) + query(rc, mid+1, e, l, r);
        }
    int ask(int a, int b, int c, int d)
        {
            int ret = 0;
            for(int i = b; i; i -= lowbit(i))
                ret += query(rt[i], 1, n, c, d);
            for(int i = a-1; i; i -= lowbit(i))
                ret -= query(rt[i], 1, n, c, d);
            return ret;
        }
    void modify(int x, int val, int k)
        {
            for(int i = x; i <= n; i += lowbit(i))
                ins(rt[i], 1, n, val, k);
        }
} t;

int main()
{
    t.init();
}
```

15 ./2-data-structure/tree-in-tree/bit-treap.cpp

```
/*
  bit 套 treap 二维数点
  func: __low(x, k)              : 统计 <=k 的数的个数
        add(x, k)                : 插入值 k
        del(x, k)                : 删除值 k
        ask(l, r, p, q)          : 二维数点 bit 中 [l, r], treap 中 [p, q]
 */
```

```
const int N = 2e5+10, M = N*3*20;
#define L(x) ch[x][0]
#define R(x) ch[x][1]
int n;
struct Treap
{
    int w[M], siz[M], ch[M][2], sz = 0, val[M];
    inline void upd(int x) {  siz[x] = siz[L(x)] + siz[R(x)] + 1; }
    void rotate(int &x, int l)
        {
            int r = 1-l, y = ch[x][l];
            ch[x][l] = ch[y][r]; ch[y][r] = x;
            upd(x); upd(y); x = y;
        }
    void __insert(int &x, int k)
        {
            if(!x) {val[x=++sz] = k; w[x] = rand(); upd(x); }
            else
            {
                int l = k > val[x];
                __insert(ch[x][l], k); upd(x);
                if(w[ch[x][l]] > w[x]) rotate(x, l);
            }
        }
    void __del(int &x, int k)
        {
            if(!x) return;
            else if(k == val[x])
            {
                if(L(x)*R(x) == 0) x = max(L(x), R(x));
                else
                {
                    int l = w[L(x)] < w[R(x)];
                    rotate(x, l); __del(ch[x][1-l], k);
                }
            }
            else __del(ch[x][k > val[x]], k);
            if(x) upd(x);
        }
    int __low(int x, int k)
        {
            int ret = 0;
            for(; x;)
            {
                if(val[x] > k) x = L(x);
                else
                {
                    ret += siz[L(x)] + 1;
                    if(val[x] == k) break;
                    x = R(x);
                }
            }
            return ret;
        }

    int rt[N];
    void add(int x, int k)
        { for(; x <= n; x += lowbit(x)) __insert(rt[x], k); }
    void del(int x, int k)
        { for(; x <= n; x += lowbit(x)) __del(rt[x], k); }
    int ask(int l, int r, int p, int q)
        {
            int ret = 0;
            for(int i = r; i; i -= lowbit(i))
                ret += __low(rt[i], q) - __low(rt[i], p-1);
            for(int i = l-1; i; i -= lowbit(i))
                ret -= __low(rt[i], q) - __low(rt[i], p-1);
            return ret;
        }
} t;
```

16 ./2-data-structure/tree/lct.cpp

```
/*
  Link Cut Tree (例为 splay 维护异或和)
  var:  sub[N]                    : splay 子树异或和
        tag[N]                    : splay 翻转标记
        prev[N]                   : path parent
  func: upd, pushdn, rotate, splay  : splay 底层操作
        modify(x, k)              : splay 中把点 x 的值改为 k
        walk(x, k)                : print 一棵 splay
        print(n)                  : print 整棵 lct(节点数量)
        pred(rt), succ(rt)        : 查询一个 splay 根的前驱/后继

        expose(x)                 : 砍掉 x 下端的链
        splice(x)                 : 将 x 所在链与 prev[x] 所在链连接
        evert(x)                  : 将 x 置为真实树的根
        access(x)                 : 将 x 到真实树的根的链抽出
        find_rt(x)                : return x 所在真实树的根
        link(u, v)                : 连接 u, v, 如果连通则 return
        cut(u, v)                 : 砍掉 e(u, v), 如果不存在则 return
        query(u, v)               : 查询 u 到 v 的路径的 sub
  note: 加点 trick: 维护边权可以把边也看成点 etc.
        在需要用到子节点前, 务必 pushdn
        修改节点信息后, 务必 upd
*/

const int N = 3e5+10;
#define L(x) ch[x][0]
#define R(x) ch[x][1]
struct LCT
{
    int sz = 0, ch[N][2], val[N], par[N], prev[N];
    int sub[N] = {0}; bool tag[N];
    inline void upd(int x) { sub[x] = sub[L(x)]^sub[R(x)]^val[x]; }
    inline void pushdn(int x)
        {
            if(!tag[x]) return;
            swap(L(x), R(x)); tag[x] = 0;
            if(L(x)) tag[L(x)] ^= 1;
            if(R(x)) tag[R(x)] ^= 1;
        }
    void rotate(int x)
        {
            // pushdn(par[x]); pushdn(x);
            int y = par[x], z = par[y], l = L(y)!=x, r = 1-l;
            if(!z) swap(prev[x], prev[y]); else ch[z][L(z)!=y] = x;
            par[x] = z; par[y] = x; par[ch[x][r]] = y;
            ch[y][l] = ch[x][r]; ch[x][r] = y;
            upd(y); upd(x);
        }
    void splay(int x)
        {
            static int stk[N]; int top = 0, cur = x;
            for(; cur; cur = par[cur]) stk[++top] = cur;
            while(top) pushdn(stk[top--]);

            while(par[x])
            {
                int y = par[x], z = par[y];
                if(par[y]) rotate((L(y)==x) ^ (L(z)==y) ?x:y);
                rotate(x);
            }
        }
    void modify(int x, int k) { splay(x); val[x] = k; upd(x); }
    void walk(int x, int dep)
        {
            if(!x) return;
            pushdn(x);
            walk(L(x), dep+1);
            rep(i, 0, dep) cout << "    ";
            cout << x << '[' << val[x] << "] p=" << par[x] << endl;
            walk(R(x), dep+1);
        }
    void print(int n)
        {
            rep(i, 1, n)
                if(!par[i])
                {
                    printf(":%d prev=%d\n", i, prev[i]);
                    walk(i, 0);
                }
        }
    int pred(int rt)
        {
            pushdn(rt); int ret = L(rt);
```

```
            while(R(ret)) ret = R(ret);
            return ret;
        }
    int succ(int rt)
        {
            pushdn(rt); int ret = R(rt);
            while(L(ret)) ret = L(ret);
            return ret;
        }

    void expose(int x)
        {
            splay(x); pushdn(x); if(!R(x)) return;
            par[R(x)] = 0; prev[R(x)] = x; R(x) = 0; upd(x);
        }
    bool splice(int x)
        {
            splay(x); if(!prev[x]) return false;
            expose(prev[x]);
            R(prev[x]) = x; par[x] = prev[x]; prev[x] = 0; upd(par[x]);
            return true;
        }
    void access(int x) { expose(x); while(splice(x)); }
    void evert(int x) { access(x); splay(x); tag[x] ^= 1; }
    int find_rt(int x)
        {
            access(x);
            do pushdn(x), x = L(x); while(L(x));
            splay(x); return x;
        }
    void link(int u, int v)
        {
            evert(u); if(find_rt(v) != u) prev[u] = v;
        }
    void cut(int u, int v)
        {
            evert(u); access(v); if(pred(v) != u) return;
            splay(v); pushdn(v); par[L(v)] = 0; L(v) = 0; upd(v);
        }
    int query(int u, int v) { evert(u); access(v); return sub[v]; }
} t;
```

17 ./2-data-structure/tree/splay.cpp

```
/*
  维护集合的 splay
  func: init()            : 初始化, 添加哨兵元素 [-1, INF]
        pred(x, k)        : x=rt, return min{ y | y <= k}
        succ(x, k)        : x=rt, return max{ y | y >= k}
        insert(w)         : 插入 w
        query(l, r)       : 查询 [l, r] 内数的个数
        del(w)          : 删除所有 w
        walk(x, dep)    : debug
  rmk:  半成品，由维护一簇集合的 splay 修改而成
 */

#define L(x) ch[x][0]
#define R(x) ch[x][1]
struct Splay
{
    int sz = 0, rt, ch[M][2], siz[M], val[M], par[M];
    int min_w[M] = {INF}, max_w[M] = {-INF};
    inline void upd(int x)
        {
            siz[x] = siz[L(x)] + siz[R(x)] + 1;
            min_w[x] = min(min(min_w[L(x)], min_w[R(x)]), val[x]);
            max_w[x] = max(max(max_w[L(x)], max_w[R(x)]), val[x]);
        }
    void init()
        {
            rt = ++sz; val[sz] = -1;
            R(rt) = ++sz; val[sz] = INF; par[sz] = rt;
            upd(sz); upd(rt);
        }

    void rotate(int x, int &k)
        {
            int y = par[x], z = par[y], l = L(y)!=x, r = 1-l;
            if(y == k) k = x; else ch[z][L(z)!=y] = x;
            par[x] = z; par[y] = x; par[ch[x][r]] = y;
            ch[y][l] = ch[x][r]; ch[x][r] = y;
            upd(y); upd(x);
        }
    void splay(int x, int &k)
        {
            static int stk[N]; int top = 0, cur = x;
            for(; cur; cur = par[cur]) stk[++top] = cur;
            while(top) upd(stk[top--]);

            while(x != k)
            {
                int y = par[x], z = par[y];
                if(y != k) rotate((L(y)==x) ^ (L(z)==y) ?x:y, k);
                rotate(x, k);
            }
        }

    int pred(int x, int k)
        {
            for(;;)
            {
                if(val[x] > k) x = L(x);
                else if(val[x] == k || min_w[R(x)] > k) return x;
                else x = R(x);
            }
        }
    int succ(int x, int k)
        {
            for(;;)
            {
                if(val[x] < k) x = R(x);
                else if(val[x] == k || max_w[L(x)] < k) return x;
                else x = L(x);
            }
        }

    void insert(int w)
        {n
            int l = pred(rt, w), r = succ(rt, w);
            splay(l, rt);
            splay(r, R(l));
            val[++sz] = w;
            L(r) = sz; par[sz] = r;
            upd(sz); upd(r); upd(l);
        }
    int query(int l, int r)
        {
            int dn = succ(rt, l), up = pred(rt, r);
            splay(dn, rt);
            if(dn == up) return 1;
            if(val[dn] > val[up]) return 0;
            splay(up, R(dn));
            return siz[L(up)] + 2;
        }
    void del(int w)
        {
            int l = pred(rt, w-1), r = succ(rt, w+1);
            splay(l, rt[x]); splay(r, R(l));
            L(r) = 0; upd(r); upd(l);
        }
    void walk(int x, int dep)
        {
            if(!x) return;
            walk(L(x), dep+1);
            rep(i, 0, dep) cout << "   ";
            cout << x << '[' << val[x] << ']' << '(' << siz[x] << ") p=" << par[x] <<
            ↪  endl;
            walk(R(x), dep+1);
        }
} t;

/*
  维护序列的 splay
  func: build(l, r, p)       : 构建区间 [l, r], 父节点为 p
        find(x, rk)          : 返回排名 rk 的元素
        reverse(l, r)        : 翻转 [l, r]
 */
```

```
#define L(x) ch[x][0]
#define R(x) ch[x][1]
struct Splay
{
    int sz = 0, rt, ch[N][2], par[N], siz[N]; bool tag[N];
    void upd(int x) { siz[x] = siz[L(x)] + siz[R(x)] + 1; }
    inline void pushdn(int x)
        {
            if(!tag[x]) return;
            swap(L(x), R(x)); tag[x] = 0;
            if(L(x)) tag[L(x)] ^= 1;
            if(R(x)) tag[R(x)] ^=1;
        }
    void rotate(int x, int &k)
        {
            pushdn(par[x]); pushdn(x);
            int y = par[x], z = par[y], l = L(y)!=x, r = 1-l;
            if(y == k) k = x; else ch[z][L(z)!=y] = x;
            par[x] = z; par[y] = x; par[ch[x][r]] = y;
            ch[y][l] = ch[x][r]; ch[x][r] = y;
            upd(y); upd(x);
        }
    void splay(int x, int &k)
        {
            static int stk[N]; int top = 0, cur = x;
            for(; cur != k; cur = par[cur]) stk[++top] = cur;
            while(top) pushdn(stk[top--]);

            while(x != k)
            {
                int y = par[x], z = par[y];
                if(y != k) rotate((L(y)==x) ^ (L(z)==y) ?x:y, k);
                rotate(x, k);
            }
        }
    int build(int l, int r, int p)
        {
            if(l > r) return 0;
            if(l == r) {par[l] = p; siz[l] = 1; return l; }
            int mid = (l+r)>>1;
            L(mid) = build(l, mid-1, mid);
            R(mid) = build(mid+1, r, mid);
            par[mid] = p; upd(mid); return mid;
        }

    int find(int x, int rk)
        {
            pushdn(x);
            if(siz[L(x)] >= rk) return find(L(x), rk);
            else if(siz[L(x)] + 1 == rk) return x;
            else return find(R(x), rk - siz[L(x)] - 1);
        }
    void reverse(int l, int r)
```

```
        {
            int u = find(rt, l-1), v = find(rt, r+1);
            splay(u, rt); splay(v, R(u)); tag[L(v)] ^= 1;
        }
    void print(int x, int tot)
        {
            if(!x) return;
            pushdn(x);
            print(L(x), tot);
            if(x != 1 && x != tot) printf("%d ", x-1);
            print(R(x), tot);
        }
} t;

int main()
{
    t.rt = t.build(1, n+2, 0);
}
```

18 ./2-data-structure/tree/treap.cpp

```
/*
  带重复计数的 treap
  var:  w[N]             : 随机值
        val[N]           : 元素值
        siz[N]           : 子树大小
        cnt[N]           : 重复值个数
        max_w/min_w[N]   : 子树最大/最小值
  func: upd(x)           : 维护 x 节点的信息
        rotate(x, l)     : 将 ch[x][l] 转到 x 的位置!
        insert(k)        : 插入 k
        del(k)           : 删除一个 k
        find(rk)         : return 排名 rk 的元素值
        rank(k)          : return k 的排名
        pred/succ(k)     : return k 的前驱/后继
 */

#define L(x) ch[x][0]
#define R(x) ch[x][1]
struct Treap
{
    int w[N], siz[N], cnt[N], ch[N][2], sz = 0, rt;
    int val[N], min_w[N] = {INF}, max_w[N] = {-INF};
    void upd(int x)
        {
            siz[x] = siz[L(x)] + siz[R(x)] + cnt[x];
            min_w[x] = min(min(min_w[L(x)], min_w[R(x)]), val[x]);
            max_w[x] = max(max(max_w[L(x)], max_w[R(x)]), val[x]);
        }
    void rotate(int &x, int l)
        {
            int r = 1-l, y = ch[x][l];
            ch[x][l] = ch[y][r]; ch[y][r] = x;
            upd(x); upd(y); x = y;
        }
    void __insert(int &x, int k)
        {
            if(!x) {val[x=++sz] = k; w[x] = rand(); cnt[x] = 1; upd(x);}
            else if(k == val[x]) {cnt[x]++; upd(x); }
            else
            {
                int l = k > val[x];
                __insert(ch[x][l], k); upd(x);
                if(w[ch[x][l]] > w[x]) rotate(x, l);
            }
        }
    void __del(int &x, int k)
        {
            if(!x) return;
            else if(k == val[x])
            {
                if(cnt[x] > 1) cnt[x]--;
                else if(L(x)*R(x) == 0) x = max(L(x), R(x));
                else
                {
                    int l = w[L(x)] < w[R(x)];
                    rotate(x, l); __del(ch[x][1-l], k);
                }
            }
            else __del(ch[x][k > val[x]], k);
            if(x) upd(x);
        }

    int __find(int x, int rk)
        {
            if(siz[L(x)] >= rk) return __find(L(x), rk);
            else if(siz[L(x)] + cnt[x] < rk)
                return __find(R(x), rk - siz[L(x)] - cnt[x]);
            else return val[x];
        }
    int __rank(int x, int k)
        {
            if(val[x] > k) return __rank(L(x), k);
            else if(val[x] == k) return siz[L(x)] + 1;
            else return siz[L(x)] + cnt[x] + __rank(R(x), k);
        }
    int __pred(int x, int k)
        {
            if(val[x] >= k) return __pred(L(x), k);
            else if(min_w[R(x)] >= k) return val[x];
            else return __pred(R(x), k);
        }
    int __succ(int x, int k)
        {
            if(val[x] <= k) return __succ(R(x), k);
            else if(max_w[L(x)] <= k) return val[x];
            else return __succ(L(x), k);
        }

    void insert(int k) { __insert(rt, k); }
    void del(int k) { __del(rt, k); }
    int find(int rk) { return __find(rt, rk); }
    int rank(int k) { return __rank(rt, k); }
    int pred(int k) { return __pred(rt, k); }
    int succ(int k) { return __succ(rt, k); }
} t;
```

19 ./2-data-structure/tree/virtual-tree.cpp

```
struct Graph
{
    struct edge{int v, w, nxt; } g[N*4];
    int head[N], sz;
    void add(int u, int v, int w)
        {
            g[++sz].v = v; g[sz].w = w;
            g[sz].nxt = head[u]; head[u] = sz;
        }
    void walk(int u, int p, int w, int dep) //debug 用
        {
            for(int i = 1; i <= dep; i++) printf("  ");
            printf("(%d)%d\n", w, u);
            for(int i = head[u]; i; i = g[i].nxt)
                if(g[i].v != p) walk(g[i].v, u, g[i].w, dep+1);
        }
} T, V;
int dep[N], dfn[N], tim = 0, par[N][20], min_c[N][20], lgn;
void dfs(int u, int p) //dfs 原树信息
{
    dep[u] = dep[p] + 1; dfn[u] = ++tim; par[u][0] = p;
    for(int i = T.head[u], v; i; i = T.g[i].nxt)
        if((v = T.g[i].v) != p) dfs(v, u), min_c[v][0] = T.g[i].w;
}
int LCA(int u, int v) //原树 lca
{
    if(dep[u] < dep[v]) swap(u, v);
    for(int i = lgn; i >= 0; i--)
        if(dep[par[u][i]] >= dep[v]) u = par[u][i];
    if(u == v) return u;
    for(int i = lgn; i >= 0; i--)
        if(par[u][i] != par[v][i])
            u = par[u][i], v = par[v][i];
    return par[u][0];
}
//根据 lca 计算 dist
```

```
int dist(int u, int v) { return dep[u]+dep[v]-2*dep[LCA(u, v)];}
int get_min(int u, int p) //虚树边信息计算 (例为边权最小值)
{
    int ret = INF;
    for(int i = lgn; i >= 0; i--)
        if(dep[par[u][i]] >= dep[p])
            ret = min(ret, min_c[u][i]), u = par[u][i];
    return ret;
}

int a[N], s[N], top = 0, vtx[N], tot = 0; //bool flg[N];
bool comp(int a, int b) { return dfn[a] < dfn[b]; }
void build()
{
    int k = read();
    for(int i = 1; i <= k; i++) flg[a[i] = read()] = 1;
    sort(a + 1, a + k + 1, comp);
    if(a[1] != 1) s[++top] = 1;
    for(int i = 1; i <= k; i++)
    {
        int cur = a[i], lca = 0/*, b = 1*/;
        while(top > 0)
        {
            lca = LCA(cur, s[top]);
            if(top > 1 && dep[lca] < dep[s[top-1]])
                V.add(s[top-1], s[top], get_min(s[top],s[top-1])),top--;
            else if(dep[lca] < dep[s[top]])
            {
                V.add(lca, s[top], get_min(s[top], lca)); top--; break;
            }
            else break;
        }
        if(lca != s[top]) s[++top] = lca;
        /*if(b)*/ s[++top] = cur;
    }
    while(top > 1)
        V.add(s[top-1], s[top], get_min(s[top], s[top-1])), top--;
}
void calc(int u, int p) //统计虚树中的点用于 clear
{
    vtx[++tot] = u;
    for(int i = V.head[u], v; i; i = V.g[i].nxt)
        if((v = V.g[i].v) != p) calc(v, u);
}
int dp(int u, int p) //注意 par[u][0] 不是 V 内父节点
{
    //...
}

int main()
{
    for(int i = 0; i <= lgn; i++) //例为维护最小边权
        for(int j = 1; j <= n; j++) min_c[j][i] = INF;
    dfs(1, 0);
    for(int i = 1; i <= lgn; i++)
        for(int j = 1; j <= n; j++)
        {
            par[j][i] = par[par[j][i-1]][i-1];
            if(par[j][i]) min_c[j][i] = min(min_c[par[j][i-1]][i-1],
                                            min_c[j][i-1]);
        }
    int m = read();
    for(int i = 1; i <= m; i++)
    {
        build();
//      V.walk(1, 0, 0, 0);
        calc(1, 0);
        printf("%lld\n", dp(1, 0));
        for(int i = 1; i <= tot; i++) //一定是 vtx[i]
            V.head[vtx[i]] /*= flg[vtx[i]]*/ = 0;
        top = tot = V.sz = 0;
    }
}

/*
  虚树小结
  1. 分清虚树和原树
     1) V or T? i or j?
     2) par[u][0] 是原树的 par
  2. clear
     1) vtx 记录节点要全 - build, dfs 过程中的剪枝会影响 vtx 的记录
        i)  dfs: if(...)return inf -> 忽略子树节点
        ii) build: if(...) break -> 在 dfs 中记录不到但 flg 有标记
     2) i 遍历 vtx 用 vtx[i] 更新!!
 */
```

20 ./3-string/acm.cpp

```
/*
  ac 自动机 (例为统计在 text 出现次数最多的 pattern 串)
  var:  N                    : pattern 数
        M                    : pattern 长度
        NM = N*M             : trie 节点数
        K                    : text 长度
        acm.vs, flg[u][c]    : 访问标号, 边 [u][c] 的访问标记
  func: init()               : init, 更新 vs
        ins(s, x)            : 向 trie 插入一个编号为 x 的 pattern
        build()              : 建 fail 指针
        query(cnt, s)        : 查询 text s, cnt 为记录答案的数组
        walk(u, c, dep)      : debug
  note: fail 树 -> fail 指针构成的树
        trie 图 -> ACm 转移规则构成的图
        u 后缀与 fail[u] 匹配, fail[u] 上的标记需要拷贝到 u 上 (查询时回溯亦可)
 */
```

```
const int N = 155, M = 75, K = 5+1E6, NM = N*M;
struct ACM
{
    int sz = 0, vs = 0, ch[NM][27], fail[NM], flg[NM][27];
    vector<int> tag[NM];
    void init() { sz = 0; vs++; }
    void ins(char *s, int x)
        {
            int u = 0;
            for(int i = 0; s[i]; i++)
            {
                int c = s[i] - 'a';
                if(flg[u][c] != vs)
                {
                    flg[u][c] = vs; ch[u][c] = ++sz;
                    fail[sz] = 0; tag[sz].clear();
                }
                u = ch[u][c];
            }
            tag[u].push_back(x);
        }
    int q[NM];
    void build()
        {
            for(int h = 0, t = 1, u; h != t; h++)
                rep(c, 0, 25)
                {
                    if(flg[u = q[h]][c] != vs) continue; // 忽略空节点
                    int v = ch[u][c];
                    if(!u) fail[v] = 0;
                    else
                    {
                        u = fail[u];
                        while(u && flg[u][c] != vs) u = fail[u];
                        int w = fail[v] = flg[u][c]==vs ? ch[u][c] : 0;
                        for(int i = 0; i < tag[w].size(); i++)
                            tag[v].push_back(tag[w][i]);
                    }
                    q[t++] = v;
                }
        }
    void query(int *cnt, char *s)
        {
            int u = 0;
            for(int i = 0; s[i]; i++)
            {
                int c = s[i] - 'a';
                if(flg[u][c] == vs) u = ch[u][c];
                else
                {
                    while(u && flg[u][c] != vs) u = fail[u];
                    u = flg[u][c]==vs ? ch[u][c] : 0;
                }
                for(int i = 0; i < tag[u].size(); i++)
                    cnt[tag[u][i]]++;
            }
        }
    void walk(int u, int c, int dep)
        {
            for(int i = 0; i < dep; i++) printf("  ");
            printf("%d(%d)(t=%d,f=%d)\n", c, u, tag[u].size(), fail[u]);
            for(int i = 0; i < 26; i++)
                if(flg[u][i] == vs) walk(ch[u][i], i, dep + 1);
        }
} acm;

int n, cnt[N]; char txt[K], pat[N][M];
int main()
{
    freopen("std.in", "r", stdin);
    while(1)
    {
        scanf("%d\n", &n); if(!n) break;
        acm.init();
        rep(i, 1, n) scanf("%s\n", pat[i]), acm.ins(pat[i], i);
        scanf("%s\n", txt);

        acm.build();
//      acm.walk(0, 0, 0);
        fill(cnt, cnt + n + 1, 0);
        acm.query(cnt, txt);

        //统计答案
        int ans = 0;
        rep(i, 1, n) ans = max(ans, cnt[i]);
        printf("%d\n", ans);
        rep(i, 1, n) if(cnt[i] == ans) printf("%s\n", pat[i]);
    }
}
```

21 ./3-string/general-sam.cpp

```
/*
  广义 sam (trie 上建 sam)
  注意节点数量
*/

const int N = 100010, CSet = 10;
struct SAM
{
    int sz;
    int len[N*2], par[N*2], ch[N*2][CSet];
    int new_node(int l)
    {
        fill(ch[sz], ch[sz]+CSet, 0);
        len[sz] = l;
        return sz++;
    }
void init()
    {
        sz = 0; new_node(0);
        par[0] = -1;
        // lst = 0;
    }

void clone_node(int p, int c, int clone)
    {
        int q = ch[p][c];
        rep(j, 0, 25) ch[clone][j] = ch[q][j];
        par[clone] = par[q];
        for(; p != -1 && ch[p][c] == q; p = par[p]) ch[p][c] = clone;
        par[q] = clone;
    }

int extend(int c, int lst)
    {
        if(ch[lst][c])
        {
            int q = ch[lst][c];
            if(len[lst] + 1 == len[q]) lst = q;
            else
            {
                int clone = new_node(len[lst]+1);
                clone_node(lst, c, clone);
                lst = clone;
            }
            return lst;
        }

        int cur = new_node(len[lst] + 1);
        int p = lst;

        for(; p != -1 && !ch[p][c]; p = par[p]) ch[p][c] = cur;
        if(p == -1) par[cur] = 0;
        else
        {
            int q = ch[p][c];
            if(len[p] + 1 == len[q]) par[cur] = q;
            else
            {
                int clone = new_node(len[p] + 1);
                clone_node(p, c, clone);
                par[cur] = clone;
            }
        }
        lst = cur;
```

```
            return lst;
        }
    void print()
        {
            rep(i, 0, sz-1)
            {
                printf("%d: len=%d, par=%d\n", (int)i, len[i], par[i]);
                rep(c, 0, 25) if(ch[i][c])
                    printf("\t -%d-> %d\n", (int)c, ch[i][c]);
            }
        }
    ll calc()
        {
            ll ret = 0;
            rep(i, 1, sz-1) ret += len[i] - len[par[i]];
            return ret;
        }

} sam;



struct Trie
{
    int ch[NM][CSet], sz, cset;
    void init(int _cset)
        {
            cset = _cset;
            sz = 0;
        }
    int insert(int pos, int c)
        {
            if(!ch[pos][c]) ch[pos][c] = ++sz;
            return ch[pos][c];
        }
    void dfs(int u, int cur)
        {
            rep(c, 0, cset-1)
                if(ch[u][c])
                {
                    int nxt = sam.extend(c, cur);
                    dfs(ch[u][c], nxt);
                }
        }
} t;

int main()
{
    sam.init();
    t.dfs(0, 0);
    printf("%lld\n", sam.calc());
}
```

22 ./3-string/hash.cpp

```
Pll operator+(const Pll&a, const Pll&b) {return mkp(a.X+b.X, a.Y+b.Y);}
Pll operator-(const Pll&a, const Pll&b) {return mkp(a.X-b.X, a.Y-b.Y);}
Pll operator*(const Pll&a, const Pll&b) {return mkp(a.X*b.X, a.Y*b.Y);}
Pll operator%(const Pll&a, const Pll&b) {return mkp(a.X%b.X, a.Y%b.Y);}
inline Pll C(ll x) { return mkp(x, x); }

const int N = 510;
Pll s[N], bn[30], B, Z;

/*
素数
61, 83, 113, 151, 211, 281, 379, 509683, 911
1217, 1627, 2179, 2909, 3881, 6907, 9209
12281, 16381, 21841, 29123, 38833, 51787, 69061, 92083
122777, 163729, 218357, 291143, 388211, 517619, 690163, 999983
1226959, 1635947, 2181271, 3877817, 5170427, 6893911, 9191891
12255871, 16341163, 29050993, 38734667, 51646229, 68861641, 91815541
1e9+7, 1e9+9
122420729, 163227661, 217636919, 290182597, 386910137, 687840301, 917120411
1222827239, 1610612741, 3221225473ul, 4294967291ul
*/
```

23 ./3-string/kmp.cpp

```
/*
  kmp: find pattern t in text s
 */
void kmp(char *s, char *t)
{
    static int p[N];
    int n = strlen(s), m = strlen(t);

    p[0] = -1; int j = -1;
    rep(i, 1, m-1)
    {
        while(j >= 0 && t[j+1] != t[i]) j = p[j];
        if(t[j+1] == t[i]) j++;
        p[i] = j;
    }

    j = -1;
    rep(i, 0, n-1)
    {
        while(j >= 0 && t[j+1] != s[i]) j = p[j];
        if(t[j+1] == s[i]) j++;
        if(j == m-1)
        {
            printf("%d\n", i-m+1);
            j = p[j];
        }
    }
}
```

24 ./3-string/manacher.cpp

```
/*
  manacher
  var:  str[N]           : 原字符串
        s[N]             : 扩充后的字符串
        rad[N]           : 回文半径
  func: manacher(n)      : manacher(字符串长度)
  note: 以 i 为中心的回文串最大长度为 f[i]-1
 */

const int N = 105;
int rad[N]; char s[N], str[N];
void manacher(int n)
{
    s[0] = '('; s[1] = '#'; s[2*n+2] = ')';
    for(int i = 0; i < n; i++) s[i*2+2] = str[i], s[i*2+3] = '#';
    n = 2*n+3;

    int r = 0, mid;
    for(int i = 1; i < n; i++)
    {
        rad[i] = i<r ? min(rad[(mid<<1)-i], rad[mid]+mid-i) : 1;
        while(s[i+rad[i]]==s[i-rad[i]]) rad[i]++;
        if(rad[i] + i > r) r = rad[i] + i, mid = i;

        // 长度 len, 在原串中中心为 center
        int len = rad[i]-1, center = i/2-1;
        if(len)
        {
            int lp = center - ceil((len-2)/2.0),
                rp = center + len/2;
            // 在原串中区间为 [lp, rp]
        }
    }
}
```

25 ./3-string/pam.cpp

```
/*
  回文自动机
  var:  ch[sz][CSet]     : 转移
        fail[sz]         : fail 指针
        len[sz]          : 节点长度
        cnt[sz]          : 节点计数
        s[n]             : 原串
        lst             : 末尾节点
  func: new_node(l)     : 新建长为 l 的节点
        init()          : 初始化 pam
        get_fail(x)     : 从 fail 链上找满足 s[n-len[x]-1] == s[n] 的节点
        insert(c)       : 向末尾插入字符 c
        calc_cnt()      : 计算每个回文串出现次数
 */

const int N = 300010, CSet = 27;
struct PAM
{
    int ch[N][CSet], fail[N], len[N], cnt[N], s[N];
    int sz, n, lst;

    int new_node(int l)
        {
            fill(ch[sz], ch[sz]+CSet, 0);
            len[sz] = l; cnt[sz] = 0;
            return sz++;
        }
    void init()
        {
            sz = 0; new_node(0); new_node(-1);
            fail[0] = 1; s[0] = -1;
            lst = n = 0;
        }
    int get_fail(int x, int pos)
        {
            while(s[pos-len[x]-1] != s[pos]) x = fail[x];
            return x;
        }
    void insert(int c)
        {
            s[++n] = c;
            int cur = get_fail(lst, n);
            if(!ch[cur][c])
            {
                int now = new_node(len[cur]+2);
                fail[now] = ch[get_fail(fail[cur], n)][c];
                ch[cur][c] = now;
            }
            cnt[lst = ch[cur][c]]++;
        }

    void print()
        {
            rep(i, 0, sz-1)
            {
                printf("%d len=%d, fail=%d\n", i, len[i], fail[i]);
                rep(j, 0, 25) if(ch[i][j])
                    printf("\t-%c-> %d\n", j+'a', ch[i][j]);
            }
        }
    void calc_cnt()
        {
            static int q[N], deg[N];
            fill(deg, deg+sz+1, 0);
            rep(i, 0, sz-1) deg[fail[i]]++;
            int h = 0, t = 0;
            rep(i, 0, sz-1) if(deg[i] == 0) q[t++] = i;

            for(; h != t; h++)
            {
                int u = q[h];
                cnt[fail[u]] += cnt[u];
                deg[fail[u]]--;
                if(deg[fail[u]] == 0) q[t++] = fail[u];
            }
        }
} pam;

char s[N];
int main()
{
    pam.init();
    for(int i = 0; s[i]; i++) pam.insert(s[i]);
    pam.calc_cnt();
}
```

26 ./3-string/sa.cpp

```
/*
  后缀数组
  var:  s[]                   : string
        rk[i]                 : rank, 保存 s[i:] 的排名
        sa[i]                 : 后缀数组, sa[rk[i]] = i
        ht[i]                 : ht[rk[i]] = LCP(s[i:], s[sa[rk[i]-1]:])
        st[][]                : ht 数组的 st 表
        lg2[x]                : log2(x)
  func: init(n)               : 初始化
        radix(str, a, b, n, m) : 基数排序 ()
        Sa(str, n, m)         : 建后缀数组 (串, 串长, 字符集大小)
        calc_ht(str, n)       : 计算 height, 建 st 表
        query(l, r)           : lcp(s[i:], s[j:]) = query(rk[i], rk[j])
 */

const int N = 1e5+10;
int lg2[N]; //floor(log2(i))
struct suffix
{
    int rk[N], sa[N], ht[N], st[N][lgN];
    void init(int n)
        {
```

```
        fill(rk, rk + n + 1, 0);
        fill(sa, sa + n + 1, 0);
        fill(ht, ht + n + 1, 0);
    }
void radix(int *str, int *a, int *b, int n, int m)
    {
        static int cnt[N];
        fill(cnt, cnt + m + 1, 0);
        rep(i, 0, n-1) cnt[str[a[i]]]++;
        rep(i, 1, m)  cnt[i] += cnt[i-1];
        per(i, n-1, 0) b[--cnt[str[a[i]]]] = a[i];
    }
void Sa(int *str, int n, int m)
    {
        static int a[N], b[N];
        rep(i, 0, n-1) rk[i] = i;
        radix(str, rk, sa, n, m); rk[sa[0]] = 0;
        rep(i, 1, n-1)
            rk[sa[i]] = rk[sa[i-1]] + (str[sa[i]]!=str[sa[i-1]]);
        for(int i = 0; (1<<i) < n; i++)
        {
            rep(j, 0, n-1)
            {
                a[j] = rk[j] + 1;
                b[j] = (j+(1<<i)>=n) ? 0 : (rk[j+(1<<i)] + 1);
                sa[j] = j;
            }
            //注意下面的字符集大小均为 n
            radix(b, sa, rk, n, n); radix(a, rk, sa, n, n);
            rk[sa[0]] = 0;
            rep(j, 1, n-1) rk[sa[j]] = rk[sa[j-1]] +
                (a[sa[j-1]]!=a[sa[j]] || b[sa[j-1]]!=b[sa[j]]);
            if(rk[sa[n-1]] == n-1) break;
        }
    }
void calc_ht(int *str, int n)
    {
        int k = 0;
        rep(i, 0, n-1)
        {
            if(rk[i] == 0) k = 0;
            else
            {
                if(k > 0) k--;
                int j = sa[rk[i]-1];
                while(i+k<n && j+k<n && str[i+k]==str[j+k]) k++;
            }
            ht[rk[i]] = k;
        }
        int lgn = ceil(log2(n));
        rep(i, 0, n-1) st[i][0] = ht[i];
        rep(j, 1, lgn) rep(i, 0, n-1)
        {
                st[i][j] = st[i][j-1]; int k = i+(1<<(j-1));
                if(k < n) st[i][j] = min(st[i][j], st[k][j-1]);
            }
        }
    int query(int l, int r)
        {
            if(l > r) swap(l, r);
            l++; int len = r-l+1, t = lg2[len];
            return min(st[l][t], st[r-(1<<t)+1][t]);
        }
} suf;

char s[N];
int main()
{
    for(int i = 0; (1<<i) < N; i++) lg2[1<<i] = i;
    for(int i = 1; i < N; i++) lg2[i] = max(lg2[i], lg2[i-1]);
    scanf("%s\n", s); int n = 0;
    for(int i = 0; s[i]; i++) str[n++] = s[i]-'0';
    suf.init(n);
    suf.Sa(str, n, 127);
    suf.clac_ht(str, n);
    //suf.query(suf.rk[i], suf.rk[j]);
}
```

27 ./3-string/sam.cpp

```
/*
  后缀自动机
  var:  sam.sz          : 节点数
        sam.lst         : 当前串末字符的节点
        sam.len         : 节点长度
        sam.par         : 后缀链接
        sam.ch          : 转移
  func: new_node(l)     : 新建一个长度 l 的节点
        init()          : 初始化 sam
        extend(c)       : 向末尾添加字符 c
        print()         : debug
        substr()        : 统计不同子串数
  note: 广义 sam 只需在做下一个串前 sam.lst = 0
 */

struct SAM
{
    int sz, lst;
    int len[N*2], par[N*2], ch[N*2][CSet];
    int new_node(int l)
        {
            fill(ch[sz], ch[sz]+CSet, 0);
            len[sz] = l;
            return sz++;
        }
    void init()
        {
            sz = 0; new_node(0);
            par[0] = -1;
            lst = 0;
        }

    void extend(int c)
        {
            int cur = new_node(len[lst] + 1);
            int p = lst;

            for(; p != -1 && !ch[p][c]; p = par[p]) ch[p][c] = cur;
            if(p == -1) par[cur] = 0;
            else
            {
                int q = ch[p][c];
                if(len[p] + 1 == len[q]) par[cur] = q;
                else
                {
                    int clone = new_node(len[p] + 1);
                    rep(j, 0, 25) ch[clone][j] = ch[q][j];
                    par[clone] = par[q];

                    for(; p != -1 && ch[p][c] == q; p = par[p])
                        ch[p][c] = clone;
                    par[q] = par[cur] = clone;
                }
            }
            lst = cur;
        }
    void print()
        {
            rep(i, 0, sz-1)
            {
                printf("%d: len=%d, par=%d\n", i, len[i], par[i]);
                rep(c, 0, 25) if(ch[i][c])
                    printf("\t -%c-> %d\n", c+'a', ch[i][c]);
            }
        }

    ll substr()
        {
            ll ret = 0;
            rep(i, 1, sz-1) ret += len[i] - len[par[i]];
            return ret;
        }
} sam;
```

28 ./4-math/algebra/fft.cpp

```
/*
  fft 多项式卷积
  var:  w/w_rev[N]          : 单位根/单位根的逆
        n/m/tot             : deg(f)/deg(g)/补全到 2^x
  func: bin_reverse(n, x[]) : 逆向二进制加法排序 x[]
        init_w(n)           : init 单位根
        fft(n, buf[], w[])  : fft(tot, 系数向量, 单位根)
  note: 注意多项式系数从 0 开始
        fft 会更改 buf 的值，多次卷积需要备份
        idft 后的答案需要/tot
 */
typedef complex<db> cpx;
const int N = 3e6+10;
void bit_reverse(int n, cpx *x)
{
    for(int i = 0, j = 0; i < n; i++)
    {
        if(i<j) swap(x[i], x[j]);
        for(int l = n >> 1; (j ^= l) < l; l >>= 1);
    }
}
void fft(int n, cpx *buf, cpx *w)
{
    bit_reverse(n, buf);
    for(int i = 2; i <= n; i <<= 1)
    {
        int m = i>>1;
        for(int j = 0; j < n; j += i)
            rep(k, 0, m-1)
            {
                cpx tmp = w[n/i*k]*buf[j+m+k];
                buf[j+m+k] = buf[j+k]-tmp; buf[j+k] += tmp;
            }
    }
}

cpx w[N], w_rev[N];
void init_w(int n)
{
    db pi = acos(-1);
    rep(k, 0, n-1)
    {
        w[k] = cpx(cos(2.0*pi*k/n), sin(2.0*pi*k/n));
        w_rev[k] = conj(w[k]);
    }
}

int n, m; cpx f[N], g[N];
int main()
{
    n = read(); m = read();
    rep(i, 0, n) { db x; scanf("%lf", &x); f[i].real(x); }
    rep(i, 0, m) { db x; scanf("%lf", &x); g[i].real(x); }
    int tot = 1<<(ll)ceil(log2(max(n, m))+1);

    init_w(tot);
    fft(tot, f, w); fft(tot, g, w); // dft
    rep(i, 0, tot-1) f[i] *= g[i];  // conj
    fft(tot, f, w_rev);             // idft
    rep(i, 0, n+m)
    {
        ll cur = round(f[i].real()/tot);
        printf("%lld ", cur);
    }
}
```

29 ./4-math/algebra/fwt.cpp

```

```

30 ./4-math/algebra/linear-basis.cpp

```
/*
  异或 (F2) 空间线性基
  var:  K                 : 最大位数
        a[N]              : data
        b[i]              : 最高位 i 的基向量
  func: gauss(int n)      : 高斯消元 (a[] 的大小)
        intersect         : 线性基求交
  note: https://blog.sengxian.com/algorithms/linear-basis
 */
const int N = 100010, K = 63;
ll a[N], b[K];
void gauss(int n)
{
    rep(i, 1, n) per(j, K-1, 0)
        if((a[i]>>j)&1)
        {
            if(b[j]) a[i] ^= b[j];
            else
            {
                b[j] = a[i];
                // 为得到各位无重复的基 O(N^3)
                per(k, j-1, 0) if(b[k] && (b[j]>>k)&1) b[j] ^= b[k];
                rep(k, j+1, K-1) if((b[k]>>j)&1) b[k] ^= b[j];
                break;
            }
        }
}
```

```
void intersect(uint *a, uint *b, uint *ans)
{
    fill(ans, ans+up, 0);
    uint c[K], d[K];
    rep(i, 0, up-1) c[i] = d[i] = b[i];
    rep(i, 0, up-1)
    {
        uint x = a[i];
        if(!x) continue;
        int j = i; uint T = 0;
        for(; j >= 0; j--)
            if((x>>j)&1)
            {
                if(c[j]) x ^= c[j], T ^= d[j];
                else break;
            }
        if(!x) ans[i] = T; else c[j] = x, d[j] = T;
    }
}
```

31 ./4-math/algebra/matrix.cpp

```
/*
  矩阵
  var:  n                     : 行数 (列数)
  func: mul(a, b, p)          : a = a*b mod p
        mat_pow(a, b, p)      : a = a^b mod p
        gauss(a)              : 解方程 a[n][n+1], 解为 a[][n+1],
                                返回系数矩阵的 det
  note: gauss 需要判断无解的情况 (出现 0 除法)
        mul, mat_pow 为 ll, gauss 为 db
 */
const int N = 105;
namespace matrix
{
    int n;
    void mul(ll a[][N], ll b[][N], ll p)
    {
        ll c[N][N];
        rep(i, 1, n) rep(j, 1, n)
        {
            c[i][j] = 0;
            rep(l, 1, n) (c[i][j] += a[i][l]*b[l][j]%p) %=p;
        }
        rep(i, 1, n) rep(j, 1, n) a[i][j] = c[i][j];
    }
    void mat_pow(ll a[][N], ll b, ll p)
    {
        ll ret[N][N];
        rep(i, 1, n) ret[i][i] = 1;
        for(; b; b >>= 1, mul(a, a, p)) if(b&1) mul(ret, a, p);
        rep(i, 1, n) rep(j, 1, n) a[i][j] = ret[i][j];
    }

    db gauss(db a[][N])
    {
        db ret = 1;
        rep(i, 1, n)
        {
            int p = i;
            rep(j, i+1, n) if(a[p][i] < a[j][i]) p = j;
            if(p != i) swap(a[p], a[i]), ret *= -1;
            ret *= a[i][i];
            rep(j, i+1, n+1) a[i][j] /= a[i][i];
            rep(j, 1, n) if(j != i)
                rep(k, i+1, n+1) a[j][k] -= a[j][i]*a[i][k];
        }
        return ret;
    }
}
```

32 ./4-math/algebra/ntt-3-mod.cpp

```
/*
  三模数模数 ntt
  var:  t[3]         : 三个不同模数的 ntt
        f,g          : 原系数向量
        fg           : 卷积后的点值表达
        p            : deg 上界
  func: NTT.calc(p, f[], g[], opt[])    : 单个 ntt 计算, 输出到 opt
        crt(x[])                        : 求三个数的 crt
 */

const __int128 Z[3] = {469762049, 998244353, 1004535809};
struct NTT
{
    ll Z, g, w[N], w_rev[N];
    NTT() {}
    void init(ll Z0, ll g0) { Z = Z0; g = g0; }
    void init_w(int n)
        {
            ll x = Pow((ll)3, (Z-1)/n, Z), y = Pow(x, Z-2, Z);
            w[0] = w_rev[0] = 1;
            rep(i, 1, n)
                (w[i] = w[i-1]*x) %=Z,
                (w_rev[i] = w_rev[i-1]*y) %=Z;
        }
    void ntt(int n, ll *buf, ll *w)
        {
            for(int i = 0, j = 0; i < n; i++)
            {
                if(i < j) swap(buf[i], buf[j]);
                for(int l = n>>1; (j^=l) < l; l >>= 1);
            }
            for(int i = 2; i <= n; i <<= 1)
            {
                int m = i>>1;
                for(int j = 0; j < n; j += i)
                    rep(k, 0, m-1)
                    {
                        ll tmp = w[n/i*k]*buf[j+m+k]%Z;
                        (buf[j+m+k] = (buf[j+k]-tmp)%Z+Z) %=Z;
                        (buf[j+k] += tmp) %=Z;
                    }
            }
        }
    void calc(int p, ll *f, ll *g, ll *opt)
        {
            static ll fy[N], gy[N];
            rep(i, 0, p-1) fy[i] = f[i], gy[i] = g[i];
            init_w(p); ntt(p, fy, w); ntt(p, gy, w);
            rep(i, 0, p-1) opt[i] = fy[i]*gy[i]%Z;
            ntt(p, opt, w_rev);

            ll r = Pow((ll)p, Z-2, Z);
            rep(i, 0, p-1) opt[i] = opt[i]*r%Z;
        }
} t[3];

__int128 crt(__int128 *x)
{
    __int128 m = Z[0]*Z[1]*Z[2];
    __int128 ans = 0;
    rep(i, 0, 2)
    {
        __int128 Mi = m/Z[i], r = Pow(Mi, Z[i]-2, Z[i]);
        (ans += Mi*r%m*x[i]%m) %=m;
    }
    return ans;
}

ll f[N], g[N], fg[3][N], ans[N]; __int128 z0;
int n, m, p;
int main()
{
    freopen("std.in", "r", stdin);
    freopen("std.out", "w", stdout);
    n = read(); m = read(); z0 = read();
    rep(i, 0, n) f[i] = read();
    rep(i, 0, m) g[i] = read();

    p = 1<<((ll)ceil(log2(max(n, m)))+1);
    rep(i, 0, 2)
    {
        t[i].init(Z[i], 3);
        t[i].calc(p, f, g, fg[i]);
        //disp(fg[i], 0, p-1);
    }

    __int128 tmp[3];
    rep(i, 0, n+m)
    {
        rep(j, 0, 2) tmp[j] = fg[j][i];
        printf("%lld ", (ll)(crt(tmp)%z0));
    }
}
```

33 ./4-math/algebra/ntt.cpp

```
/*
  快速数论变换 (% 2^k+1)
  var:  w/w_rev[N]          : 单位根/单位根的逆
        n/m/tot             : deg(f)/deg(g)/次数上界
  func: init_w(n)           : init 单位根
        ntt(n, buf[], w[])  : ntt(tot, 系数向量, 单位根)
  note: 注意多项式系数从 0 开始
        ntt 会更改 buf 的值, 多次卷积需要备份
        idnt 后的答案乘逆元!
 */

const int N = 2e6+10; const ll Z = 998244353;
ll w[N], w_rev[N];
void init_w(int n)
{
    ll x = Pow(3, (Z-1)/n, Z), y = Pow(x, Z-2, Z);
    w[0] = w_rev[0] = 1;
    rep(i, 1, n)
        (w[i] = w[i-1]*x) %=Z,
        (w_rev[i] = w_rev[i-1]*y) %=Z;
}

void ntt(int n, ll *buf, ll *w)
{
    for(int i = 0, j = 0; i < n; i++)
    {
        if(i < j) swap(buf[i], buf[j]);
        for(int l = n>>1; (j^=l) < l; l >>= 1);
    }
    for(int i = 2; i <= n; i <<= 1)
    {
        int m = i>>1;
        for(int j = 0; j < n; j += i)
            rep(k, 0, m-1)
            {
                ll tmp = w[n/i*k]*buf[j+m+k]%Z;
                (buf[j+m+k] = (buf[j+k]-tmp)%Z+Z) %=Z;
                (buf[j+k] += tmp) %=Z;
            }
        }
}

int n, m; ll f[N];
int main()
{
    freopen("std.in", "r", stdin);
    //freopen("std.out", "w", stdout);
    n = read(); m = read();
    rep(i, 1, m) f[read()] = 1;
    int tot = 1<<(ll)ceil(log2(5*n)+1); // 例为多项式次幂

    init_w(tot); ntt(tot, f, w);             // 正变换
    rep(i, 0, tot) f[i] = Pow(f[i], n/2, Z); // do something
    ntt(tot, f, w_rev);                      // 逆变换
    ll x = Pow(tot, Z-2, Z);
    rep(i, 0, tot) (f[i]*=x) %=Z;            // 除 tot^-1
}
```

34 ./4-math/combinatorics/matrix-tree.cpp

```
/*
  Kirchhoff's theorem
  生成树个数 = 拉普拉斯矩阵 L 的 |V|-1 阶主子式的行列式
  l(u, v) = u 的度数               u==v
            -1 * u 和 v 间的边数   u!=v
  变元 kirchhoff: 生成树边权积的和 = ... 行列式

  例为 luogu p3317: 给出每条边连通的概率，求恰好生成一棵树的概率
  https://www.luogu.org/problemnew/solution/P3317
 */
#include<bits/stdc++.h>
using namespace std;
typedef long double ld;
const ld EPS = 1e-15; const int N = 55;
inline bool equ(ld a, ld b) { return abs(a-b)<EPS; }

ld a[N][N];
int gauss(int n)
{
    int ret = 1;
    for(int i = 1; i <= n; i++)
    {
        int l = i;
        for(int j = i+1; j <= n; j++) if(a[l][i]<a[j][i]) l = j;
        if(l != i) swap(a[l], a[i]), ret *= -1;
        for(int j = i+1; j <= n; j++)
            for(int k = i+1; k <= n; k++)
                a[j][k] -= a[j][i]*a[i][k]/max(EPS, a[i][i]);
    }
    return ret;
}

int main()
{
    freopen("std.in", "r", stdin);
    int n; scanf("%d\n", &n); ld tmp = 1;
    for(int i = 1; i <= n; i++)
        for(int j = 1; j <= n; j++)
        {
            ld x; scanf("%Lf", &x);
            if(i == j) continue;
            if(j > i) tmp *= max(EPS, 1-x);
            a[i][j] = -x/max(EPS, 1-x);
        }
    for(int i = 1; i <= n; i++)
        for(int j = 1; j <= n; j++)
            if(i != j) a[i][i] -= a[i][j];
    // for(int i = 1; i <= n; i++, printf("\n"))
    //  for(int j = 1; j <= n; j++) printf("%0.2Lf\t", a[i][j]);

    ld ans = gauss(n-1)*tmp;
    for(int i = 1; i <= n-1; i++) ans *= a[i][i];
    printf("%0.8Lf", ans);
}
```

35 ./4-math/combinatorics/polya.cpp

```
/*
  环形 polya: m 种颜色给 n 个珠子的项链涂色
  burnside: 轨道数 * |G| = sum{ f(g) | any g in G }
  polya:    f(g) = m^c(g)
  环形: 涂色数 * n = sum{ phi(k) * m^(n/k), k|d }
  带翻转的话就是 Dn, 翻转分奇偶讨论即可

  var:  p[sz], cnt[sz]               : n 的质因数分解
        ans                          : 答案
  func: factor(n)                    : 质因分解 n
        dfs(x, val, phi)             : dfs 计算答案 (位置, 值, phi 值)
 */
const int N = 1e6+10; const ll Z = 1e9+7;
ll p[N], cnt[N], sz = 0;
void factor(ll n)
{
    sz = 0;
    for(ll i = 2; i*i <= n; i++)
        if(n%i == 0)
        {
            p[++sz] = i; cnt[sz] = 0;
            while(n%i == 0) cnt[sz]++, n /= i;
        }
    if(n != 1) p[++sz] = n, cnt[sz] = 1;
}

ll n, m, ans = 0;
void dfs(int x, ll val, ll phi)
{
    // cout << x << ": " << val << ", " << phi << endl;
    if(x > sz)
    {
        ll y = n/val;
        ll tmp = Pow(m, y, Z);
        (ans += phi * tmp % Z) %= Z;
        // printf("phi[%d]=%d\n", val, phi);
        return;
    }
    dfs(x+1, val, phi);
    ll cur = p[x];
    rep(i, 1, cnt[x])
    {
        ll new_phi = phi * (p[x]-1) * cur / p[x];
        dfs(x+1, val * cur, new_phi);
        cur *= p[x];
    }
}

int main()
{
    freopen("std.in", "r", stdin);
    // freopen("std.out", "w", stdout);
    while(scanf("%lld%lld", &n, &m) != EOF)
    {
        factor(n);
        ans = 0; dfs(1, 1, 1);
        // cout << n << ", " << m << ":" << ans << endl;
        printf("%lld\n", ans * Pow(n, Z-2, Z) % Z);
    }
}
```

36 ./4-math/misc/permutation.cpp

```
/*
  排列相关算法
  func:  merge(b, l, r)     : return 逆序数 (归并排序)
 */

int merge(int *b, int l, int r)
{
    static int c[N];
    if(l>=r) return 0;
    int mid = (l+r)>>1;
    int ret = merge(b, l, mid) + merge(b, mid+1, r);
    int p = l, q = mid+1;
    rep(i, l, r)
    {
        if(p>mid || (q<=r && b[q]<b[p])) c[i] = b[q++], ret += mid-p+1;
        else c[i] = b[p++];
    }
    rep(i, l, r) b[i] = c[i];
    return ret;
}
```

37 ./4-math/number-theory/crt.cpp

```
/*
  中国剩余定理
  var:  求最小非负的 x  x = a[i] (mod m[i]) 1<=i<=n
  func: extgcd(a, b, x, y)      : 扩展欧几里得
        inv(a, b)               : 求 a 的 mod b 下的逆元
        crt()                   : 普通 crt (m[i] 两两互素)
        extcrt()                : 扩展 crt (无解返回-1)
 */

typedef __int128 big;
const int N = 1e5+10;
big a[N], m[N]; int n;

big extgcd(big a, big b, big &x, big &y)
{
    if(!b) { x = 1; y = 0; return a; }
    big d = extgcd(b, a%b, y, x);
    y -= (a/b)*x;
    return d;
}

big inv(big a, big b)
{
    big x, y; extgcd(a, b, x, y);
    return (x%b+b)%b;
}

big crt()
{
    big M = 1, ans = 0;
    rep(i, 1, n) M *= m[i];
    rep(i, 1, n) (ans += a[i]*(M/m[i])*inv(M/m[i], m[i])) %=M;
    return (ans+M)%M;
}

bool calc(int i)
{
    // x = a[i] + m[i]*y1 = a[i+1] - m[i+1]*y2
    //     m[i]*y1 + m[i+1]*y2 = a[i+1] - a[i];
    big y1, y2, d = extgcd(m[i], m[i+1], y1, y2);
    if((a[i+1]-a[i]) % d) return 0;
    big b0 = a[i] + (a[i+1]-a[i])/d*y1*m[i],
        a0 = m[i]/d*m[i+1];
    b0 = (b0%a0+a0)%a0;
    m[i+1] = a0; a[i+1] = b0;
    return 1;
}

big extcrt()
{
    rep(i, 1, n-1) if(!calc(i)) return -1;
    big x, y; extgcd(1, m[n], x, y);
    return (x*a[n]%m[n] + m[n])%m[n];
}
```

38 ./4-math/number-theory/discrete-log.cpp

```
/*
  求满足 x^k=a (mod p) 的所有 x, 不存在输出-1
  (sgu 261) k * ind(x) = ind(a) (mod phi(p))
            求 ind(x) 即可
  var:  p, k, a                    : 参数
        fac[N], sz                 : p 的因子, 栈顶指针
  func: extgcd(a, b, &x, &y)       : extgcd
        get_rt()                   : 求 p 的原根 g
        bsgs(g)                    : 求满足 g^y=a (mod p) 的 y (求 ind(a))
        solv(p, k, a)              : 接口
 */

namespace disc_log
{
    ll extgcd(ll a, ll b, ll &x, ll &y)
    {
        if(!b) { x = 1; y = 0; return a; }
        ll d = extgcd(b, a%b, y, x);
        y -= (a/b)*x;
        return d;
    }

    ll p, k, a, fac[N], sz = 0;
    ll get_rt()
    {
        ll n = p-1;
        for(int i = 2; i*i <= n; i++)
            if(n%i == 0)
            {
                fac[++sz] = i;
                while(n%i == 0) n/=i;
            }
        if(n != 1) fac[++sz] = n;
        n = p-1;
        rep(i, 1, n)
        {
            bool flg = 0;
            rep(j, 1, sz) if(Pow(i, n/fac[j], p) == 1) { flg = 1; break; }
            if(!flg) return i;
        }
    }

    map<ll, ll> hs;
    ll bsgs(ll g)
    {
        ll q = ceil(sqrt(p)), cur = 1;
        rep(i, 0, q-1) hs.insert(mkp(a*cur%p, i)), (cur*=g)%=p;
        ll tmp = Pow(g, q, p); cur = 1;
        rep(i, 0, q)
        {
            map<ll, ll>::iterator it = hs.find(cur);
            if(it != hs.end()) return q*i-(*it).Y;
            (cur*=tmp)%=p;
        }
        printf("-1"); exit(0);
    }

    void solv(ll k0, ll a0, ll p0)
    {
        a = a0; k = k0; p = p0; ll g = get_rt();

        ll t = bsgs(g);
        ll phi = p-1, x0, y0, d = extgcd(k, phi, x0, y0);
        if(t%d) { printf("-1"); return; }

        ll step = phi/d;
        x0 = x0*t/d%step;
        if(x0 < 0) x0 += step;
        printf("%lld", Pow(g, x0, p));
    }
}
```

39 ./4-math/number-theory/euler.cpp

```
/*
  欧拉函数相关
  var:  phi[]           : 欧拉函数
        p[sz], flg[]    : 素数, 合数标记
  func: euler(n)        : 线性筛 euler 函数
 */

const int N = 1e7+10;
int phi[N], p[N], sz = 0; bool flg[N];
```

```
void euler(int n)
{
    phi[1] = 1;
    rep(i, 2, n)
    {
        if(!phi[i]) phi[i] = i-1, p[++sz] = i;
        for(int j = 1; i*p[j] <= n; j++)
            if(i%p[j]) phi[i*p[j]] = phi[i]*phi[p[j]];
            else { phi[i*p[j]] = phi[i]*p[j]; break; }
    }
}

/*
  euler 定理: (a, m) = 1 => a^phi[m] = 1 (mod m)
  扩展 euler: a^q = a^{ q mod phi[m] + phi[m] } (mod m)
 */
```

40 ./4-math/number-theory/lucas.cpp

```
/*
  lucas 定理 求 C(x, k)%p
  var:  fac[k]                  : k!%p
  func: extgcd(a, b, x, y)      : 扩展欧几里得
        inv(a, b)               : 求 a 的 mod b 下的逆元
        C(x, k)                 : C(x, k)%p

  x = sum { a_i * P^i }
  k = sum { b_i * P^i }
  C(x, k)%P = prod { C(a_i, b_i)%P }
 */

ll fac[N], p;
void extgcd(ll a, ll b, ll &x, ll &y)
{
    if(!b) { x = 1; y = 0; return; }
    extgcd(b, a%b, y, x);
    y -= (a/b)*x;
}

ll inv(ll a, ll b)
{
    ll x, y; extgcd(a, b, x, y);
    return (x%b+b)%b;
}

inline ll __C(int x, int k)
{return (fac[x]*inv(fac[k])%p*inv(fac[x-k])%p+p)%p;}
ll C(ll x, ll k)
{
    ll ret = 1;
    for(; x; x /= p, k /= p)
    {
        ll a = x%p, b = k%p;
        if(a < b) return 0;
        ret = ret*__C(a, b)%p;
    }
    return ret;
}

int main()
{
    fac[0] = 1; rep(i, 1, n) fac[i] = (fac[i-1]*i)%p;
}
```

41 ./4-math/number-theory/mobius.cpp

```
/*
  mobius 相关
  var:  mu[]                  : mobius 函数
        p[sz], flg[]          : 素数, 合数标记
  func: calc_mu(n)            : 线性筛 mobius
        calc_val(n)           : 数论分块
        calc_pref(n)          : 杜教筛
 */

const int N = 50010;
ll mu[N], p[N], sz = 0; bool flg[N];
void calc_mu(int n)
{
    mu[1] = 1;
    rep(i, 2, n)
    {
        if(!flg[i]) p[++sz] = i, mu[i] = -1;
        rep(j, 1, sz)
        {
            if(p[j]*i > 5e4) break;
            flg[p[j]*i] = 1;
            if(i%p[j]) mu[i*p[j]] = mu[i]*mu[p[j]];
            else { mu[i*p[j]] = 0; break; }
        }
    }
}

/*
  calc_val(n) = sum { floor(n/i) * f(i) | 1 <= i <= n }
  F(x) 为 f(x) 的前缀和
 */
ll calc_val(ll n)
{
    ll ans = 0;
    for(ll i = 1, j; i <= n; i = j + 1)
    {
        j = n/(n/i);
        ans += (n/i) * (F(j) - F(i-1));
    }
    return ans;
}

/*
  calc_pref(n) = sum { f(i) | 1<=i<=n } 即 f 的前缀和
  h = f*g, H 为 h 的前缀和, G 为 g 的前缀和

  up = N^(2/3)
  g1 = g(1)
 */
ll up, f[N], g1; map<ll, ll> mp;
int calc_pref(ll n) {
    if(n <= up) return pref_f[n];
    auto it = mp.find(n); if(it != mp.end()) return it->Y;
    int ans = H(n);
    for (ll i = 2, j; i <= n; i = j + 1)
    {
        j = n/(n/i);
        ans -= (G(j) - G(i-1)) * calc_pref(n/i);
    }
    ans /= g1;
    return mp[n] =  ans;
}
```

42 ./4-math/number-theory/modsqr.cpp

```
ll modsqr(ll a, ll n)
{
    ll b, k, i, x;
    a = (a%n + n)%n;
    // printf("a=%lld\n", a);
    if(a == 0) return 0;
    if(n == 2) return a%n;
    if(Pow(a, (n-1)/2, n) == 1)
    {
        if(n%4 == 3) x = Pow(a, (n+1)/4, n);
        else
        {
            for(b = 1; Pow(b, (n-1)/2, n) == 1; b++);
            i = (n-1)/2;
            k = 0;
            do
            {
                i /= 2;
                k /= 2;
                if((Pow(a, i, n)*Pow(b, k, n)+1)%n == 0) k += (n-1)/2;
            }
            while(i%2 == 0);
            x = Pow(a, (i+1)/2, n) * Pow(b, k/2, n) % n;
        }
        if(x*2 > n) x = n - x;
        return x;
    }
    return -1;
}
```

43 ./5-graph-theory/2-sat.cpp

```
/*
  2-SAT: Solve CNF formula (x1 or y1) and (x2 or y2) ... and (xn or yn)
  (x1 or y1) == (!x1 -> y1) and (!y1 -> x1)

  求强连通分量，若任意!a 和 a 不在一个分量内则可行
  拓扑序在前的赋值为 false
 */
```

44 ./5-graph-theory/euler-tour.cpp

```
/*
  无向图欧拉序 O(E)
  var:  deg[]           : 度数
        seq[top]        : 存放求出的 euler 序
  func: euler(n)        : 欧拉序 (点数 n) return 是否存在
  note: 这里利用了 head 指针递增的 trick，保证每条边只被遍历一次
 */

const int N = 2e5+10;
struct edge{ int v, nxt; bool flg; } g[N*2];
int head[N], sz = 1;
void add(int u, int v)
{
    g[++sz].v = v; g[sz].flg = 1;
    g[sz].nxt = head[u]; head[u] = sz;
}

int deg[N];
int seq[N], top = 0;
void dfs(int u)
{
    // printf(" -> %d\n", u);
    while(head[u])
    {
        int i = head[u];
        head[u] =  g[i].nxt;
        if(g[i].flg)
        {
            // printf("%d", u);
            g[i].flg = g[i^1].flg = 0;
            dfs(g[i].v);
        }
    }
    seq[++top] = u;
}

bool euler(int n)
{
    vector<int> sta;
    rep(i, 1, n) if(deg[i]%2) sta.pb(i);
    if(sta.size() == 2) dfs(sta[0]);
    else if(sta.size() == 0) dfs(1);
    else return 0;

    if(top != n) return 0; else return 1;
}
```

45 ./5-graph-theory/hungary.cpp

```
/*
  二分图最大匹配
  var:  mat[N]       : 匹配节点, 缺省值 0
        flg[N]       : 访问标记
        vs           :  version 标记 (用于检查 flg)
  func: dfs(u)       : 增广 (u)
        hungary()    : return 最大匹配数
  note: 其实 mat 只需要记录一边即可
        注意 mat 会和 matrix 变量冲突

  rmk:  二分图
        1) 最大匹配数 = 最小点覆盖
        2) 最少边覆盖 = 点数-最大匹配数 = 最大独立集
        DAG
        1) 最少不相交路径覆盖: V 拆成 (Vx, Vy), e(u,v) 变成 Uy->Vx
           = 原图点数 - 最大匹配
        2) 最小可相交: 先 floyd 后 1)
        3) 最大独立集 = 最少不相交路径覆盖 (dilworth)
 */

int mat[N], flg[N], vs = 1;
bool dfs(int u)
{
    for(int i = head[u], v; i; i = g[i].nxt)
        if(flg[v = g[i].v] != vs)
        {
            flg[v] = vs;
            if(!mat[v] || dfs(mat[v]))
            {
                mat[u] = v; mat[v] = u;
                return true;
            }
        }
    return false;
}
int hungary()
{
    int cnt = 0;
    rep(i, 1, n) { vs++; if(dfs(i)) cnt++;}
    return cnt;
}
```

46 ./5-graph-theory/km.cpp

```
/*
  km (二分图最大权匹配) O(N^3)
  var:  g[N][N]              : g[u][v] = x_u 和 y_v 间的边 !
        vis_x/vis_y[N]       : 访问标记
        lx/ly[N]             : 顶标 (lx[u]+ly[v]>=g[u][v])
        slack[N]             : slack[v] = min{lx[u]+ly[v]-g[u][v]}
        mat[N]               : Y 集合中节点的匹配
  func: dfs(u)               : 增广 (u) return 是否成功
        km()                 : 最大权完备匹配
  note: 这里的 g 不是邻接矩阵 (行标为 x 集合, 列标为 y 集合)
        不完备匹配可以通过补 0 边和空节点得到
        最小权匹配将不等式反向, slack 换成 max 即可 ?
        稠密图下表现良好, 稀疏图的效率不如费用流
 */

bool vis_x[N], vis_y[N];
ll g[N][N], lx[N], ly[N], slack[N]; int mat[N];
bool dfs(int u)
{
    vis_x[u] = 1;
    rep(v, 1, n)
    {
        if(vis_y[v]) continue;
        ll tmp = lx[u]+ly[v]-g[u][v];
        if(tmp == 0)
        {
            vis_y[v] = 1;
            if(mat[v] == -1 || dfs(mat[v]))
            {  mat[v] = u; return true; }
        } else slack[v] = min(slack[v], tmp);
    }
    return false;
}

ll km()
{
    rep(i, 1, n) rep(j, 1, n) lx[i] = max(lx[i], g[i][j]);
    fill(ly+1, ly+n+1, 0); fill(mat+1, mat+n+1, -1);
    rep(u, 1, n)
    {
        fill(slack, slack+n+1, INF);
        for(;;)
        {
            fill(vis_x, vis_x+n+1, 0); fill(vis_y, vis_y+n+1, 0);
            if(dfs(u)) break;
            ll delta = INF;
            rep(v, 1, n) if(!vis_y[v]) delta = min(delta, slack[v]);
            rep(i, 1, n) if(vis_x[i]) lx[i] -= delta;
            rep(i, 1, n)
                if(vis_y[i]) ly[i] += delta; else slack[i] += delta;
        }
    }
    ll ret = 0;
    rep(v, 1, n) ret += g[mat[v]][v];
    return ret;
}
```

47 ./5-graph-theory/lca.cpp

```
const int N = 2e5+10, lgN = 20;
struct edge { int v, nxt; } g[N*2];
int head[N], sz = 0, w[N], loc[N];
void add(int u, int v) { g[++sz] = {v, head[u]}; head[u] = sz; }

int n, lgn;
namespace binary
{
    int par[N][lgN], dep[N];
    void dfs(int u, int p)
    {
        dep[u] = dep[p] + 1; par[u][0] = p;
        for(int i = head[u], v; i; i = g[i].nxt)
            if((v = g[i].v) != p) dfs(v, u);
    }
    void init()
    {
        dfs(1, 0);
        lgn = ceil(log2(n));
        rep(i, 1, lgn) rep(j, 1, n) par[j][i] = par[par[j][i-1]][i-1];
    }

    int lca(int u, int v)
    {
        if(dep[u] < dep[v]) swap(u, v);
        per(i, lgn, 0) if(dep[par[u][i]] >= dep[v]) u = par[u][i];
        if(u == v) return u;
        per(i, lgn, 0)
            if(par[u][i] != par[v][i]) u = par[u][i], v = par[v][i];
        return par[u][0];
    }
}

namespace st
{
    int lg2[2*N], st[2*N][20], dep[N], dfn[N], ord[2*N], tot = 0;
    void dfs(int u, int p)
    {
        dep[u] = dep[p] + 1; ord[++tot] = u; dfn[u] = tot;
        for(int i = head[u], v; i; i = g[i].nxt)
            if((v = g[i].v) != p) dfs(v, u), ord[++tot] = u;
    }
    int Min(int a, int b) { return dep[a]<dep[b] ? a : b; }
    void init()
    {
        lgn = ceil(log2(n));
        for(int i = 0; (1<<i) <= n; i++) lg2[1<<i] = i;
        rep(i, 1, n) lg2[i] = max(lg2[i], lg2[i-1]);
        rep(i, 1, n) st[i][0] = ord[i];
        rep(j, 1, lgn) rep(i, 1, n)
        {
            st[i][j] = st[i][j-1]; int k = i+(1<<(j-1));
            if(k < n) st[i][j] = Min(st[i][j], st[k][j-1]);
        }
    }

    int lca(int u, int v)
    {
        int l = dfn[u], r = dfn[v]; if(l > r) swap(l, r);
        int len = r-l+1, t = lg2[len];
        return Min(st[l][t], st[r-(1<<t)+1][t]);
    }

    bool ischd(int x, int p) { return lca(x, p) == p; }
    bool onpath(int l, int x, int r)
    {
        int p = lca(l, r);
        if(p != l && p != r) return onpath(l, x, p) || onpath(p, x, r);
        if(dep[l] < dep[r]) swap(l, r);
        return ischd(l, x) && ischd(x, r);
    }
}
```

48 ./5-graph-theory/max-clique.cpp

```
/*
  无向图最大团 (折半枚举)
  var:  n              : 顶点数
        g[u]           : 点 u 的相邻点的 bitmask
        f[s]           : 子图 s 的最大团
        all, fir, sec  : all = 2^n-1, fir = 2^(n/2)-1, sec = all^fir
  func: clique()       : 返回最大团大小
  note: 枚举 sec 中的状态 s0, calc(s0) 计算包含 s0 的最大团数
 */

const int N = 1e5+10, M = 42, K = (ll)1<<20;
int n; ll g[N]; int f[K];
int dp(ll s)
{
    if(!s) return 0;
    if(f[s] != -1) return f[s];
    f[s] = 0; ll v = lowbit(s), idx = round(log2(v));
    return f[s] = max(dp(s^v), dp(g[idx]&s)+1);
}

ll fir, sec, all;
int calc(ll s0)
{
    ll sta = all, cnt = 0;
    for(ll s = s0; s; s -= lowbit(s))
    {
        ll v = lowbit(s), idx = round(log2(v));
        sta &= g[idx]|v; cnt++;
    }
    if((s0|sta) != sta) return 0;
    return dp(fir&sta) + cnt;
}

ll clique()
{
    int ans = 0, hlf = n/2;
    all = ((ll)1<<n)-1, fir = ((ll)1<<hlf)-1, sec = all^fir;
    fill(f, f + ((ll)1<<hlf), -1);
    for(ll i = sec; i; i = (i-1)&sec) ans = max(ans, calc(i));
    return ans;
}
```

49 ./5-graph-theory/max-flow.cpp

```
int q[N], d[N];
bool bfs(int s, int dest)
{
    fill(d, d + dest + 1, INF); d[s] = 0; q[0] = s;
    for(int h = 0, t = 1, u, v; h != t; h = (h+1)%N)
        for(int i = head[u = q[h]]; i; i = g[i].nxt)
            if(d[v = g[i].v] == INF && g[i].cap)
                d[v] = d[u] + 1, q[t++] = v, t %= N;
    return d[dest] != INF;
}
int dfs(int u, int t, int f)
{
    if(u == t) return f;
    int ret = 0;
    for(int i = head[u], v; i; i = g[i].nxt)
        if(g[i].cap && d[v = g[i].v] == d[u] + 1) //忘判 g[i].cap
        {
            int tmp = dfs(v, t, min(f - ret, g[i].cap));
            g[i].cap -= tmp; g[i^1].cap += tmp; ret += tmp;
            if(!tmp) d[v] = INF; //勿忘
            if(ret == f) return ret;
        }
    return ret;
}
int mf(int s, int t)
{
    int ret = 0;
    while(bfs(s, t)) ret += dfs(s, t, INF);
    return ret;
}
```

50 ./5-graph-theory/min-cost-max-flow.cpp

```
/*
  最小费用最大流
  var:  q[N], d[N], inq[N]       : spfa
        pre[i]                   : s->i 的最短路上连向 i 的边
  func: spfa(int s, int dest)    : 找增广路, return 是否存在增广路
        mcmf(int s, int t)       : min_cost_max_flow(源点, 汇点)
  note: mcmf(s, t) t 须为最大节点
 */

struct edge { int, v, w, nxt, cap; } g[N*2];
int sz = 1, head[N];
void __add(int u, int v, int cap, int w)
{
    g[++sz].nxt = head[u]; head[u] = sz;
    g[sz].w = w; g[sz].v = v; g[sz].cap = cap;
}
void add(int u, int v, int cap, int w)
{ __add(u, v, cap, w); __add(v, u, 0, -w); }

int q[N], d[N], pre[N]; bool inq[N];
bool spfa(int s, int dest)
{
    fill(d, d + dest + 1, INF);
    fill(pre, pre + dest + 1, 0);
    q[0] = s; d[s] = 0; inq[s] = 1;
    for(int h = 0, t = 1, u, v; h != t; h = (h+1)%N, inq[u] = 0)
        for(int i = head[u = q[h]]; i; i = g[i].nxt)
            if(g[i].cap && d[v = g[i].v] > d[u] + g[i].w)
            {
                d[v] = d[u] + g[i].w; pre[v] = i;
                if(!inq[v]) inq[v] = 1, q[t++] = v, t %= N;
            }
    return d[dest] != INF;
}
int mcmf(int s, int t)
{
    int ret = 0;
    while(spfa(s, t))
    {
        int tmp = INF, sum = 0;
        for(int i = pre[t]; i; i = pre[g[i^1].v])
            tmp = min(tmp, g[i].cap), sum += g[i].w;
        ret += tmp*sum;
        for(int i = pre[t]; i; i = pre[g[i^1].v])
            g[i].cap -= tmp, g[i^1].cap += tmp;
    }
    return ret;
}
```

51 ./5-graph-theory/sp.cpp

```
// dijkstra
bool used[N];
typedef pair<ll, int> Pli;
void dij(int s, ll *d)
{
    priority_queue<Pli, vector<Pli>, greater<Pli> > q;
    fill(used, used + n + 1, 0);
    fill(d, d + n + 1, INF);
    d[s] = 0; q.push(make_pair((ll)0, s));
    while(!q.empty())
    {
        int u = q.top().second; q.pop();
        if(used[u]) continue; used[u] = 1;
        for(int i = head[u], v; i; i = g[i].nxt)
            if(d[v = g[i].v] > d[u] + g[i].w)
            {
                d[v] = d[u] + g[i].w;
                q.push(make_pair(d[v], v));
            }
    }
}

// spfa
int q[N]; bool inq[N];
void spfa(int s, ll *d)
{
    fill(inq, inq+n+2, false);
    fill(d, d+n+2, INF);
    d[s] = 0; inq[s] = true; q[0] = s;
    int u, v;
    for(int h = 0, t = 1; h!=t; inq[q[h]] = false, h = (h+1)%N)
        for(int i = head[u = q[h]]; i; i = g[i].nxt)
```

```
                if(d[v = g[i].v]>d[u]+g[i].w)
                {
                    d[v] = d[u]+g[i].w;
                    if(!inq[v]) q[t] = v, inq[v] = true, t = (t+1)%N;
                }
}


// floyd
void floyd()
{
    //要初始化
    for(int k = 0; k < n; k++)
        for(int i = 0; i < n; i++)
            for(int j = 0; j < n; j++)
                f[i][j] = min(f[i][j], f[i][k]+f[k][j]);
}

//路径输出的 floyd; g[i][j] 为 i->j 的第一个节点
int f[N][N], g[N][N];
void floyd()
{
    for(int k = 1; k <= n; k ++)
        for(int i = 1; i <= n; i ++)
            for(int j = 1, cur; j <= n; j ++)
                if(f[i][j] >= (cur = f[i][k] + f[k][j])) //字典序最小
                {
                    if(f[i][j] == cur && g[i][j] <= g[i][k]) continue;
                    f[i][j] = cur; g[i][j] = g[i][k];
                }
}

// floyd 求最小环
int min_circuit()
{
    int ans = INF;
    rep(k, 1, n)
    {
        rep(i, 1, k-1) rep(j, i+1, k-1)
            ans = min(ans, f[i][j] + g[i][k] + g[k][j]);
        rep(i, 1, n) rep(j, 1, n)
            f[i][j] = min(f[i][j], f[i][k] + f[k][j]);
    }
    return ans;
}
```

52 ./5-graph-theory/tarjan.cpp

```
//强连通分量
int cc = 0, tim = 0, t = 0, scc[N], dfn[N], low[N], s[N], loc[N];
bool ins[N], vis[N];
```

```
void tarjan(int u)
{
    dfn[u] = low[u] = ++tim;
    ins[u] = vis[u] = 1; s[loc[u] = t++] = u;
    for(int i = head[u]; i; i = g[i].nxt)
    {
        int v = g[i].v;
        if(!vis[v]) tarjan(v), low[u] = min(low[u], low[v]);
        else if(ins[v]) low[u] = min(low[u], dfn[v]);
    }
    if(dfn[u] == low[u])
    {
        cc++;
        while(t != loc[u]) scc[s[--t]] = cc, ins[s[t]] = 0;
    }
}


//双连通分量
/*
关于割点
  1. 由于是无向图，不需要 ins 的判断：如果可达必定 ins
     -> 若不需要标记分量则不用入栈
  2. rt 需要单独判断
  3. * 割点不可以出栈
  4. 缩点后是一颗黑白染色的树
关于桥
  1. 桥可以将图划分为不相交的连通块，割点不可以
  2. 注意反向边的问题
 */
int n, m; int tim = 0, low[N], dfn[N];
//int stk[N], top = 0;
void tarjan(int u, int p)
{
    dfn[u] = low[u] = ++tim; //stk[++top] = u;
    int chd = 0; //特判 rt(默认为 1)
    for(int i = head[u], v; i; i = g[i].nxt)
        if((v = g[i].v) != p)
            if(!dfn[v])
            {
                tarjan(v, u); chd++; siz[u] += siz[v];
                low[u] = min(low[v], low[u]);
                if((u==1 && chd>1) || (u!=1 && low[v] >= dfn[u]))
                    solv(u); //u 为割点，u 不可以出栈
                if(low[v] > dfn[u])
                    solv(i); //i 为桥，u 不可出栈
            }
            else low[u] = min(dfn[v], low[u]);
    //如果只求边双连通分量，同 scc
    //if(low[u]==dfn[u]) solv(u);
}
```

53 ./6-geometry/basics.cpp

```
/*
  点，直线，浮点修正
  func:      int cmp(double)                  判断 double 符号
             double rec(double, double)       四舍五入 (避免负 0)
             det(point, point)                叉积
             dot(point, point)                点积
             dist(point, point)               距离
  class:     point                            点
             line                             线段 (直线)
  mathod:    line:p_dist(point)               点到直线距离
             line:p_on(point)                 点是否在线段上
 */

int cmp(double x)
{
    if(fabs(x) < EPS) return 0;
    if(x > 0) return 1; else return -1;
}

inline double rec(double x, double prec)
{
    x = round(x*prec)/prec;
    return sgn(x) == 0 ? 0 : x;
}

struct point
{
    double x, y;
    point(){}
    point(double a, double b) : x(a), y(b) {}
    void input() { scanf("%lf%lf", &x, &y); }
    friend point operator + (const point &a, const point &b)
        { return point(a.x+b.x, a.y+b.y); }
    friend point operator - (const point &a, const point &b)
        { return point(a.x-b.x, a.y-b.y); }
    friend point operator *  (const double &a, const point &b)
        { return point(a*b.x, a*b.y); }
    double norm() { return sqrt(squ(x)+squ(y)); }
};

double det(const point &a, const point &b) { return a.x*b.y-a.y*b.x; }
double dot(const point &a, const point &b) { return a.x*b.x+a.y*b.y; }
double dist(const point &a, const point &b) { return (a-b).norm(); }

struct line
{
    point a, b;
    line() {};
    line(point x, point y): a(x), b(y) {}
    double p_dist(point x) { return fabs(det(x-a, x-b) / dist(a, b)); }
    bool p_on(point x)
        {
            return cmp(det(x-a, x-b)) == 0 && cmp(dot(x-a, x-b)) <= 0;
        }
};
```

54 ./6-geometry/convex.cpp

```
/*
  凸包
  func:     convex_hull(vector<point>)           求凸包 (不保留共线点)
            convex_hull_stable(vector<point>)    求凸包 (保留共线点)
            ploy_line_intersect::init            将凸包边的极角排序
            ploy_line_intersect::solv            O(logN) 判断凸包与直线相交
  method:   convex:p_in                          O(logN) 判断点在凸包内
            convex:area                          面积
            convex:p_in                          旋转卡壳 (不允许共线点)
 */

#define succ(i) ((i+1)%n)
#define S(i, j, k) det(a[j]-a[i], a[k]-a[i])
struct convex
{
    point a[N]; int n;
    convex(int k = 0): n(k) {}
    int p_in(point b)
        {
            point g = (a[0] + a[n/3] + a[2*n/3]) / 3.;
            int l = 0, r = n;
            while(r - l > 1)
            {
                int mid = (l+r)>>1;
                if(cmp(det(a[l]-g, a[mid]-g)) > 0)
                {
                    if(cmp(det(a[l]-g, b-g))>=0 &&
                       cmp(det(a[mid]-g, b-g)<0)) r = mid;
                    else l = mid;
                }
                else
                {
                    if(cmp(det(a[l]-g, b-g))<0 &&
                       cmp(det(a[mid]-g, b-g))>=0) l = mid;
                    else r = mid;
                }
            }
            r %= n;
            return cmp(det(a[l]-b, a[r]-b));
        }
    double area()
        {
            double ret = 0;
            rep(i, 0, n-1) ret += det(a[i+1], a[i]);
            return abs(ret/2.);
        }

    // 不可以有共线点
    double diameter()
        {
            if(n == 1) return 0;
            double ret = 0;
            int j = 1;
            rep(i, 0, n-1)
            {
                while(cmp(S(i, succ(i), j)-S(i, succ(i), succ(j)))<0)
                    j = succ(j);
                ret = max(ret, dist(a[i], a[j]));
                ret = max(ret, dist(a[succ(i)], a[succ(j)]));
            }
            return ret;
        }
};

bool comp_h(const point &a, const point &b)
{
    return cmp(a.x-b.x)<0 || (cmp(a.x-b.x)==0 && cmp(a.y-b.y)<0);
}

convex convex_hull_stable(vector<point> a)
{
    convex ret = convex(2*a.size()+5);
    sort(a.begin(), a.end(), comp_h);
    a.erase(unique(a.begin(), a.end()), a.end());

    int m = 0;
    rep(i, 0, a.size()-1)
    {
        while(m>1 && cmp(det(ret.a[m-1]-ret.a[m-2], a[i]-ret.a[m-1]))<0)
            m--;
        ret.a[m++] = a[i];
    }
    int k = m;
    per(i, a.size()-1, 0)
    {
        while(ret.a[m-1]==a[i] ||
              (m>k&&(cmp(det(ret.a[m-1]-ret.a[m-2], a[i]-ret.a[m-1]))<0)))
            m--;
        ret.a[m++] = a[i];
    }
    ret.n = m - (a.size()!=1);
    return ret;
}


convex convex_hull(vector<point> a)
```

```
{
    convex ret = convex(2*a.size()+5);
    sort(a.begin(), a.end(), comp_h);
    a.erase(unique(a.begin(), a.end()), a.end());

    int m = 0;
    rep(i, 0, (int)a.size()-1)
    {
        while(m>1 && cmp(det(ret.a[m-1]-ret.a[m-2], a[i]-ret.a[m-1]))<=0)
            m--;
        ret.a[m++] = a[i];
    }
    int k = m;
    per(i, (int)a.size()-1, 0)
    {
        while(m>k&&(cmp(det(ret.a[m-1]-ret.a[m-2], a[i]-ret.a[m-1]))<=0))
            m--;
        ret.a[m++] = a[i];
    }
    ret.n = m - (a.size()!=1);
    return ret;
}



namespace ploy_line_intersect
{
    double angle(point p) { return atan2(p.y, p.x); }
    int n; vector<point> p;
    pair<double, int> seq[N];
    void init(convex &ploy)
    {
        // 这里的 n 是凸包的点数
        rep(i, 0, n-1) seq[i] = mkp(angle(ploy.a[i+1]-ploy.a[i]), i);
        sort(seq, seq+n);
        seq[n] = seq[0];
    }

    bool solv(convex &ploy, point s, point e)
    {
        int i = upper_bound(seq, seq+n, mkp(angle(e-s), 0)) - seq,
            j = upper_bound(seq, seq+n, mkp(angle(s-e), 0)) - seq;
        point p1 = ploy.a[seq[i].Y], p2 = ploy.a[seq[j].Y];
        return cmp(det(p1-s, e-s) * det(p2-s, e-s)) < 0;
    }
}
```

55 ./6-geometry/half-plane.cpp

```
/*
  半平面
  class:    plane                        半平面 (表示 s->e 左侧)
  func:     hpi(plane[], int)            半平面交
            convex cut(convex, plane)    半平面和多边形交
            core(ploygon)                多边形的核
  method:   plane:&                      直线求交
 */

struct plane
{
    point s,e; // 线段左侧的半平面
    double k;
    plane(){}
    plane(point _s,point _e)
        {
            s = _s; e = _e;
            k = atan2(e.y-s.y, e.x-s.x);
        }
    point operator &(const plane &b) const
        {
            point res = s;
            double t = det(s-b.s, b.s-b.e) / det(s-e, b.s-b.e);
            res.x += (e.x-s.x)*t;
            res.y += (e.y-s.y)*t;
            return res;
        }
};

bool hpi_cmp(const plane &a, const plane &b)
{
    if(fabs(a.k-b.k) > EPS) return a.k < b.k;
    return det(a.s-b.s, b.e-b.s) < 0;
}

plane q[N];
convex hpi(line ps[], int n)
{
    int tot = n;
    sort(ps, ps + n, hpi_cmp);

    tot = 1;
    rep(i, 1, n-1)
        if(fabs(ps[i].k - ps[i-1].k) > EPS) ps[tot++] = ps[i];
    int h = 0, t = 1;
    q[0] = ps[0]; q[1] = ps[1];
    rep(i, 2, tot-1)
    {
        if(fabs(det(q[t].e - q[t].s, q[t-1].e - q[t-1].s)) < EPS ||
           fabs(det(q[h].e - q[h].s, q[h+1].e - q[h+1].s)) < EPS)
            return convex(0);
        while(h < t &&
              det((q[t]&q[t-1])-ps[i].s, ps[i].e-ps[i].s) > EPS) t--;
        while(h < t &&
              det((q[h]&q[h+1])-ps[i].s, ps[i].e-ps[i].s) > EPS) h++;
        q[++t] = ps[i];
    }
    while(h < t &&
          det((q[t]&q[t-1])-q[h].s, q[h].e-q[h].s) > EPS) t--;
    while(h < t &&
          det((q[h]&q[h-1])-q[t].s, q[t].e-q[t].e) > EPS) h++;
    if(t <= h+1) return convex(0);

    convex ret(0);
    rep(i, h, t-1) ret.pb(q[i] & q[i+1]);
    if(h < t-1) ret.pb(q[h] & q[t]);
    return ret;
}

#define succ(i) ((i+1)%n)
#define pred(i) (i?i-1:n-1)
convex cut(convex &ploy, plane &L)
{
    int n = ploy.n;
    convex ret;
    rep(i, 0, n-1)
        if(cmp(L.calc(ploy.a[i])) < 0) ret.pb(ploy.a[i]);
        else
        {
            if(cmp(det(L.e-L.s, pred(i)-L.s)) > 0)
                ret.pb(L & plane(ploy.a[pred(i)], ploy.a[i]));
            if(cmp(det(L.e-L.s, succ(i)-L.s)) > 0)
                ret.pb(L & plane(ploy.a[i], ploy.a[succ(i)]));
        }
    return ret;
}

// ploy 逆时针
convex core(polygon &ploy)
{
    convex ret;
    ret.push_back(point(-INF, -INF));
    ret.push_back(point(INF, -INF));
    ret.push_back(point(INF, INF));
    ret.push_back(point(-INF, INF));
    int n = ploy.n;
    rep(i, 0, n-1)
    {
        plane L(ploy.a[i], ploy.a[succ(i)]);
        ret = cut(ret, L);
    }
    return ret;
}
```

56 ./6-geometry/ploygon.cpp

```
/*
  多边形类
  method:   input(int)                          读入多边形 (逆时针)
            p_in(point)                         判断点在形内 (环顾法)
            area()                              面积
            mass_center()                       重心
            border_int_p()                      边上整点数
            inside_int_p()                      形内整点数
 */

struct polygon
{
    int n; point a[N];
    polygon(){}
    void input(int k)
        {
            n = k;
            rep(i, 0, n-1) a[i].input();
            a[n] = a[0];
        }
    int p_in(point t)
        {
            double sum = 0;
            rep(i, 0, n-1)
            {
                if(line(a[i], a[i+1]).p_on(t)) return 0;
                int sgn = cmp(det(a[i]-t, a[i+1]-t));
                double theta = acos(dot(a[i]-t, a[i+1]-t) /
                                    (dist(a[i], t) * dist(a[i+1], t)));
                sum += theta * sgn;
            }
            return fabs(sum) > PI/2 ? 1 : -1;
        }
    double area()
        {
            double ret = 0;
            rep(i, 0, n-1) ret += det(a[i+1], a[i]);
            return ret/2.;
        }
    point mass_center()
        {
            point ret = point(0, 0);
            if(cmp(area()) == 0) return ret;
            rep(i, 0, n-1) ret = ret + det(a[i+1], a[i])*(a[i]+a[i+1]);
            return ret/area()/6.;
        }

    int border_int_p()
        {
            int ret = 0;
            rep(i, 0, n-1)
            {
                point tmp = a[i+1]-a[i];
                ret += abs(gcd(int(tmp.x), int(tmp.y)));
            }
            return ret;
        }
    int inside_int_p() { return int(area()-border_int_p()/2.+1); }
};
```

57 ./7-misc/discrete.cpp

```
/*
  离散化
  var:  a[N]          : 所有值
        b[N]          : 待离散化的数组, size 为 n
  func: discrete()    : 接口
*/

const int N = 1e5;
int n, b[N]; vector<int> a;
void discrete()
{
    sort(a.begin(), a.end());
    a.erase(unique(a.begin(), a.end()), a.end());
    rep(i, 1, n) b[i] = lbound(a.begin(), a.end(), b[i]) - a.begin();
}
```

58 ./7-misc/mt19937.cpp

```
mt19937 rng32((ll)time(0));
int myrand(int i) { return rng32()%i; }
```