# Standard Code Library

Song of Midnight Clock

Xiamen University

October 2022

# Contents

# 一切的开始

## 代码头

- hkk 版

```cpp
#include <bits/stdc++.h>
using namespace std;

#define fec(i, x, y) (int i = head[x], y = g[i].to; i; i = g[i].ne, y = g[i].to)
#define dbg(...) fprintf(stderr, __VA_ARGS__)
#define fi first
#define se second

using ll = long long; using ull = unsigned long long; using pii = pair<int, int>;

template <typename A, typename B> bool smax(A &a, const B &b) { return a < b ? a = b, 1 : 0; }
template <typename A, typename B> bool smin(A &a, const B &b) { return b < a ? a = b, 1 : 0; }
```

# 图论建图模板

## 带权图

```cpp
struct Edge {int to, ne, w;} g[M]; int head[N], tot;
void addedge(int x, int y, int z) { g[++tot].to = y; g[tot].w = z; g[tot].ne = head[x]; head[x] = tot; }
void adde(int x, int y, int z) { addedge(x, y, z); addedge(y, x, z); }
```

## 无权图

```cpp
struct Edge {int to, ne;} g[M]; int head[N], tot;
void addedge(int x, int y) { g[++tot].to = y; g[tot].ne = head[x]; head[x] = tot; }
void adde(int x, int y) { addedge(x, y); addedge(y, x); }
```

## 网络流

```cpp
struct Edge {int to, ne, f;} g[M * 2]; int head[N], tot = 1;
void addedge(int x, int y, int z) { g[++tot].to = y; g[tot].f = z; g[tot].ne = head[x]; head[x] = tot; }
void adde(int x, int y, int z) { addedge(x, y, z); addedge(y, x, 0); }
```

# 取模及组合数

## 取模

```cpp
int smod(int x) { return x >= P ? x - P : x; }
void sadd(int &x, const int &y) { x += y; x >= P ? x -= P : x; }
int fpow(int x, int y) {
    int ans = 1;
    for (; y; y >>= 1, x = (ll)x * x % P) if (y & 1) ans = (ll)ans * x % P;
    return ans;
}
```

## 组合数

```cpp
int fac[N], inv[N], ifac[N];
void ycl(const int &n = ::n) {
    fac[0] = 1; for (int i = 1; i <= n; ++i) fac[i] = (ll)fac[i - 1] * i % P;
    inv[1] = 1; for (int i = 2; i <= n; ++i) inv[i] = (ll)(P - P / i) * inv[P % i] % P;
    ifac[0] = 1; for (int i = 1; i <= n; ++i) ifac[i] = (ll)ifac[i - 1] * inv[i] % P;
}
int C(int x, int y) {
    if (x < y) return 0;
    return (ll)fac[x] * ifac[y] % P * ifac[x - y] % P;
}
```

## 快速读写

### 普通的快速度入

```cpp
template<typename I> inline void read(I &x) {
    int f = 0, c;
    while (!isdigit(c = getchar())) c == '-' ? f = 1 : 0;
    x = c & 15;
    while (isdigit(c = getchar())) x = (x << 1) + (x << 3) + (c & 15);
    f ? x = -x : 0;
}
```

### fread/fwrite

```cpp
namespace io {
    const int SIZE = (1 << 21) + 1;
    char ibuf[SIZE], *iS, *iT, obuf[SIZE], *oS = obuf, *oT = obuf + SIZE - 1, c, qu[55];
    int f, qr;
    #define gc() (iS == iT ? (iT = (iS = ibuf) + fread(ibuf, 1, SIZE, stdin), iS == iT ? EOF : *(iS++)) : *(iS++))
    void flush() { fwrite(obuf, 1, oS - obuf, stdout), oS = obuf; }
    void putc(char c) { *(oS++) = c; if (oS == oT) flush(); }
    template <typename I> void read(I& x) {
        for (f = 1, c = gc(); c < '0' || c > '9'; c = gc()) c == '-' ? f = -1 : 0;
        for (x = 0; c >= '0' && c <= '9'; c = gc()) x = (x << 1) + (x << 3) + (c & 15);
        ~f ? 0 : x = -x;
    }
    template <typename I> void write(I x) {
        if (x == 0) putc('0');
        if (x < 0) x = -x, putc('-');
        for (; x; x /= 10) qu[++qr] = x % 10 + '0';
        for (; qr;) putc(qu[qr--]);
    }
    struct Flusher_ { ~Flusher_() { flush(); } } io_flusher;
}
```

## 对拍

```bash
#!/usr/bin/env bash
g++ -o r main.cpp -O2 -std=c++17
g++ -o std std.cpp -O2 -std=c++17
while true; do
    python gen.py > in
    ./std < in > stdout
    ./r < in > out
    if test $? -ne 0; then
        exit 0
    fi
    if diff stdout out; then
        printf "AC\n"
    else
        printf "GG\n"
        exit 0
    fi
done
```

- 快速编译运行（配合无插件 VSC）

```bash
#!/bin/bash
g++ $1.cpp -o $1 -O2 -std=c++14 -Wall -Dzerol -g
if $? -eq 0; then
    ./$1
fi
```

# 数据结构

## ST 表

- 二维

```cpp
int f[maxn][maxn][10][10];
inline int highbit(int x) { return 31 - __builtin_clz(x); }
inline int calc(int x, int y, int xx, int yy, int p, int q) {
    return max(
        max(f[x][y][p][q], f[xx - (1 << p) + 1][yy - (1 << q) + 1][p][q]),
        max(f[xx - (1 << p) + 1][y][p][q], f[x][yy - (1 << q) + 1][p][q])
    );
}
void init() {
    FOR (x, 0, highbit(n) + 1)
    FOR (y, 0, highbit(m) + 1)
        FOR (i, 0, n - (1 << x) + 1)
        FOR (j, 0, m - (1 << y) + 1) {
            if (!x && !y) { f[i][j][x][y] = a[i][j]; continue; }
            f[i][j][x][y] = calc(
                i, j,
                i + (1 << x) - 1, j + (1 << y) - 1,
                max(x - 1, 0), max(y - 1, 0)
            );
        }
}
inline int get_max(int x, int y, int xx, int yy) {
    return calc(x, y, xx, yy, highbit(xx - x + 1), highbit(yy - y + 1));
}
```

- 一维

```cpp
int f[N][LOG];
void init() {
    for (int i = 1; i <= n; ++i) f[i][0] = a[i];
    for (int j = 1; j < LOG; ++j)
        for (int i = 1; i + (1 << j) - 1 <= n; ++i)
            f[i][j] = max(f[i][j - 1], f[i + (1 << (j - 1))][j - 1]);
}
int getmax(int l, int r) {
    int k = log2(r - l + 1); // 使用 g++ 编译可以使用 __lg
    return max(f[l][k], f[r - (1 << k) + 1][k]);
}
```

### 线段树

- 普适

```cpp
namespace sg {
    struct Q {
        LL setv;
        explicit Q(LL setv = -1): setv(setv) {}
        void operator += (const Q& q) { if (q.setv != -1) setv = q.setv; }
    };
    struct P {
        LL min;
        explicit P(LL min = INF): min(min) {}
        void up(Q& q) { if (q.setv != -1) min = q.setv; }
    };
    template<typename T>
    P operator & (T&& a, T&& b) {
        return P(min(a.min, b.min));
    }
    P p[maxn << 2];
    Q q[maxn << 2];
#define lson o * 2, l, (l + r) / 2
#define rson o * 2 + 1, (l + r) / 2 + 1, r
    void up(int o, int l, int r) {
        if (l == r) p[o] = P();
        else p[o] = p[o * 2] & p[o * 2 + 1];
        p[o].up(q[o]);
    }
    void down(int o, int l, int r) {
        q[o * 2] += q[o]; q[o * 2 + 1] += q[o];
        q[o] = Q();
        up(lson); up(rson);
```

```
    }
    template<typename T>
    void build(T&& f, int o = 1, int l = 1, int r = n) {
        if (l == r) q[o] = f(l);
        else { build(f, lson); build(f, rson); q[o] = Q(); }
        up(o, l, r);
    }
    P query(int ql, int qr, int o = 1, int l = 1, int r = n) {
        if (ql > r || l > qr) return P();
        if (ql <= l && r <= qr) return p[o];
        down(o, l, r);
        return query(ql, qr, lson) & query(ql, qr, rson);
    }
    void update(int ql, int qr, const Q& v, int o = 1, int l = 1, int r = n) {
        if (ql > r || l > qr) return;
        if (ql <= l && r <= qr) q[o] += v;
        else {
            down(o, l, r);
            update(ql, qr, v, lson); update(ql, qr, v, rson);
        }
        up(o, l, r);
    }
}
```

- SET + ADD

```
struct IntervalTree {
#define ls o * 2, l, m
#define rs o * 2 + 1, m + 1, r
    static const LL M = maxn * 4, RS = 1E18 - 1;
    LL addv[M], setv[M], minv[M], maxv[M], sumv[M];
    void init() {
        memset(addv, 0, sizeof addv);
        fill(setv, setv + M, RS);
        memset(minv, 0, sizeof minv);
        memset(maxv, 0, sizeof maxv);
        memset(sumv, 0, sizeof sumv);
    }
    void maintain(LL o, LL l, LL r) {
        if (l < r) {
            LL lc = o * 2, rc = o * 2 + 1;
            sumv[o] = sumv[lc] + sumv[rc];
            minv[o] = min(minv[lc], minv[rc]);
            maxv[o] = max(maxv[lc], maxv[rc]);
        } else sumv[o] = minv[o] = maxv[o] = 0;
        if (setv[o] != RS) { minv[o] = maxv[o] = setv[o]; sumv[o] = setv[o] * (r - l + 1); }
        if (addv[o]) { minv[o] += addv[o]; maxv[o] += addv[o]; sumv[o] += addv[o] * (r - l + 1); }
    }
    void build(LL o, LL l, LL r) {
        if (l == r) addv[o] = a[l];
        else {
            LL m = (l + r) / 2;
            build(ls); build(rs);
        }
        maintain(o, l, r);
    }
    void pushdown(LL o) {
        LL lc = o * 2, rc = o * 2 + 1;
        if (setv[o] != RS) {
            setv[lc] = setv[rc] = setv[o];
            addv[lc] = addv[rc] = 0;
            setv[o] = RS;
        }
        if (addv[o]) {
            addv[lc] += addv[o]; addv[rc] += addv[o];
            addv[o] = 0;
        }
    }
    void update(LL p, LL q, LL o, LL l, LL r, LL v, LL op) {
        if (p <= r && l <= q)
        if (p <= l && r <= q) {
            if (op == 2) { setv[o] = v; addv[o] = 0; }
```

```
            else addv[o] += v;
        } else {
            pushdown(o);
            LL m = (l + r) / 2;
            update(p, q, ls, v, op); update(p, q, rs, v, op);
        }
        maintain(o, l, r);
    }
    void query(LL p, LL q, LL o, LL l, LL r, LL add, LL& ssum, LL& smin, LL& smax) {
        if (p > r || l > q) return;
        if (setv[o] != RS) {
            LL v = setv[o] + add + addv[o];
            ssum += v * (min(r, q) - max(l, p) + 1);
            smin = min(smin, v);
            smax = max(smax, v);
        } else if (p <= l && r <= q) {
            ssum += sumv[o] + add * (r - l + 1);
            smin = min(smin, minv[o] + add);
            smax = max(smax, maxv[o] + add);
        } else {
            LL m = (l + r) / 2;
            query(p, q, ls, add + addv[o], ssum, smin, smax);
            query(p, q, rs, add + addv[o], ssum, smin, smax);
        }
    }
} IT;
```

## zkw 线段树

- 单点加，区间和/max/min

```
namespace zkw_segment_tree{
    const int inf=0x7fffffff;
    const int N=262143;
    //set to 2^{logn}-1

    int T1[N<<1|10],T2[N<<1|10],T3[N<<1|10];
    //Çø¼ä°Í£¬Çø¼ämax£¬Çø¼ämin

    void modify(int x,int v) {
        for(T1[x+=N]+=v,T2[x]+=v,T3[x]+=v;x>>=1;) {
            T1[x]+=v;
            T2[x]=std::max(T2[x<<1],T2[x<<1|1]);
            T3[x]=std::min(T3[x<<1],T3[x<<1|1]);
        }
    }

    int query1(int l,int r) {
        int ret=0;
        for(l+=N-1,r+=N+1;l^r^1;l>>=1,r>>=1) {
            if(l&1^1) ret+=T1[l^1];
            if(r&1) ret+=T1[r^1];
        }
        return ret;
    }

    int query2(int l,int r) {
        int ret=0;
        for(l+=N-1,r+=N+1;l^r^1;l>>=1,r>>=1) {
            if(l&1^1) ret=std::max(T2[l^1],ret);
            if(r&1) ret=std::max(T2[r^1],ret);
        }
        return ret;
    }

    int query3(int l,int r) {
        int ret=inf;
        for(l+=N-1,r+=N+1;l^r^1;l>>=1,r>>=1) {
            if(l&1^1) ret=std::min(T3[l^1],ret);
            if(r&1) ret=std::min(T3[r^1],ret);
        }
```

```
        return ret;
    }
}
```

### 李超线段树

- 维护平面中若干条线段，查询某横坐标下纵坐标最高的线段编号

```
struct Seg {
    int x0, x1;
    double  k, b;
} seg[N + 1];
int tot;
namespace lctree
{
int tr[(M << 2) + 1];
inline double inter(int x, int y)
{
    return (seg[y].b - seg[x].b) / (seg[x].k - seg[y].k);
}
inline double calc(int num, int x)
{
    return seg[num].k * x + seg[num].b;
}
inline void update(int now, int l, int r, int num)
{
    if (l >= seg[num].x0 && r <= seg[num].x1) {
        if (!tr[now]) {
            tr[now] = num;
            return;
        }
        double al = calc(tr[now], l), ar = calc(tr[now], r), bl = calc(num, l), br = calc(num, r);
        if (bl <= al && br <= ar)
            return;
        if (bl >= al && br >= ar) {
            tr[now] = num;
            return;
        }
        double w = inter(tr[now], num);
        if (bl >= al) {
            if (w <= mid) {
                update(lson, l, mid, num);
            } else {
                update(rson, mid + 1, r, tr[now]);
                tr[now] = num;
            }
        } else {
            if (w <= mid) {
                update(lson, l, mid, tr[now]);
                tr[now] = num;
            } else {
                update(rson, mid + 1, r, num);
            }
        }
    } else {
        if (mid >= seg[num].x0)
            update(lson, l, mid, num);
        if (mid < seg[num].x1)
            update(rson, mid + 1, r, num);
    }
    return;
}
inline int query(int now, int l, int r, int pos)
{
    int ans = tr[now], x;

    if (l == r)
        return ans;
    if (pos <= mid)
        x = query(lson, l, mid, pos);
    else
```

```
        x = query(rson, mid + 1, r, pos);
    if (calc(x, pos) >= calc(ans, pos))
        ans = x;
    return ans;
}
}
```

### 均摊复杂度线段树

- 区间取 min，区间求和。

```
namespace R {
#define lson o * 2, l, (l + r) / 2
#define rson o * 2 + 1, (l + r) / 2 + 1, r
    int m1[N], m2[N], cm1[N];
    LL sum[N];
    void up(int o) {
        int lc = o * 2, rc = lc + 1;
        m1[o] = max(m1[lc], m1[rc]);
        sum[o] = sum[lc] + sum[rc];
        if (m1[lc] == m1[rc]) {
            cm1[o] = cm1[lc] + cm1[rc];
            m2[o] = max(m2[lc], m2[rc]);
        } else {
            cm1[o] = m1[lc] > m1[rc] ? cm1[lc] : cm1[rc];
            m2[o] = max(min(m1[lc], m1[rc]), max(m2[lc], m2[rc]));
        }
    }
    void mod(int o, int x) {
        if (x >= m1[o]) return;
        assert(x > m2[o]);
        sum[o] -= 1LL * (m1[o] - x) * cm1[o];
        m1[o] = x;
    }
    void down(int o) {
        int lc = o * 2, rc = lc + 1;
        mod(lc, m1[o]); mod(rc, m1[o]);
    }
    void build(int o, int l, int r) {
        if (l == r) { int t; read(t); sum[o] = m1[o] = t; m2[o] = -INF; cm1[o] = 1; }
        else { build(lson); build(rson); up(o); }
    }
    void update(int ql, int qr, int x, int o, int l, int r) {
        if (r < ql || qr < l || m1[o] <= x) return;
        if (ql <= l && r <= qr && m2[o] < x) { mod(o, x); return; }
        down(o);
        update(ql, qr, x, lson); update(ql, qr, x, rson);
        up(o);
    }
    int qmax(int ql, int qr, int o, int l, int r) {
        if (r < ql || qr < l) return -INF;
        if (ql <= l && r <= qr) return m1[o];
        down(o);
        return max(qmax(ql, qr, lson), qmax(ql, qr, rson));
    }
    LL qsum(int ql, int qr, int o, int l, int r) {
        if (r < ql || qr < l) return 0;
        if (ql <= l && r <= qr) return sum[o];
        down(o);
        return qsum(ql, qr, lson) + qsum(ql, qr, rson);
    }
}
```

### 持久化线段树

- ADD

```
namespace tree {
#define mid ((l + r) >> 1)
#define lson ql, qr, l, mid
```

```cpp
#define rson ql, qr, mid + 1, r
    struct P {
        LL add, sum;
        int ls, rs;
    } tr[maxn * 45 * 2];
    int sz = 1;
    int N(LL add, int l, int r, int ls, int rs) {
        tr[sz] = {add, tr[ls].sum + tr[rs].sum + add * (len[r] - len[l - 1]), ls, rs};
        return sz++;
    }
    int update(int o, int ql, int qr, int l, int r, LL add) {
        if (ql > r || l > qr) return o;
        const P& t = tr[o];
        if (ql <= l && r <= qr) return N(add + t.add, l, r, t.ls, t.rs);
        return N(t.add, l, r, update(t.ls, lson, add), update(t.rs, rson, add));
    }
    LL query(int o, int ql, int qr, int l, int r, LL add = 0) {
        if (ql > r || l > qr) return 0;
        const P& t = tr[o];
        if (ql <= l && r <= qr) return add * (len[r] - len[l - 1]) + t.sum;
        return query(t.ls, lson, add + t.add) + query(t.rs, rson, add + t.add);
    }
}
```

## K-D Tree

**最优化问题一定要用全局变量大力剪枝，而且左右儿子先递归潜力大的**

- 维护信息
- 带重构（适合在线）
- 插入时左右儿子要标记为 `null`。

```cpp
namespace kd {
    const int K = 2, inf = 1E9, M = N;
    const double lim = 0.7;
    struct P {
        int d[K], l[K], r[K], sz, val;
        LL sum;
        P *ls, *rs;
        P* up() {
            sz = ls->sz + rs->sz + 1;
            sum = ls->sum + rs->sum + val;
            FOR (i, 0, K) {
                l[i] = min(d[i], min(ls->l[i], rs->l[i]));
                r[i] = max(d[i], max(ls->r[i], rs->r[i]));
            }
            return this;
        }
    } pool[M], *null = new P, *pit = pool;
    static P *tmp[M], **pt;
    void init() {
        null->ls = null->rs = null;
        FOR (i, 0, K) null->l[i] = inf, null->r[i] = -inf;
        null->sum = null->val = 0;
        null->sz = 0;
    }

    P* build(P** l, P** r, int d = 0) { // [l, r)
        if (d == K) d = 0;
        if (l >= r) return null;
        P** m = l + (r - l) / 2; assert(l <= m && m < r);
        nth_element(l, m, r, [&](const P* a, const P* b){
            return a->d[d] < b->d[d];
        });
        P* o = *m;
        o->ls = build(l, m, d + 1); o->rs = build(m + 1, r, d + 1);
        return o->up();
    }
    P* Build() {
        pt = tmp; FOR (it, pool, pit) *pt++ = it;
```

```
        return build(tmp, pt);
    }
    inline bool inside(int p[], int q[], int l[], int r[]) {
        FOR (i, 0, K) if (r[i] < q[i] || p[i] < l[i]) return false;
        return true;
    }
    LL query(P* o, int l[], int r[]) {
        if (o == null) return 0;
        FOR (i, 0, K) if (o->r[i] < l[i] || r[i] < o->l[i]) return 0;
        if (inside(o->l, o->r, l, r)) return o->sum;
        return query(o->ls, l, r) + query(o->rs, l, r) +
               (inside(o->d, o->d, l, r) ? o->val : 0);
    }
    void dfs(P* o) {
        if (o == null) return;
        *pt++ = o; dfs(o->ls); dfs(o->rs);
    }
    P* ins(P* o, P* x, int d = 0) {
        if (d == K) d = 0;
        if (o == null) return x->up();
        P*& oo = x->d[d] <= o->d[d] ? o->ls : o->rs;
        if (oo->sz > o->sz * lim) {
            pt = tmp; dfs(o); *pt++ = x;
            return build(tmp, pt, d);
        }
        oo = ins(oo, x, d + 1);
        return o->up();
    }
}
```

- 维护信息
- 带修改（适合离线）

```
namespace kd {
    const int K = 3, inf = 1E9, M = N << 3;
    extern struct P* null;
    struct P {
        int d[K], l[K], r[K], val;
        int Max;
        P *ls, *rs, *fa;
        P* up() {
            Max = max(val, max(ls->Max, rs->Max));
            FOR (i, 0, K) {
                l[i] = min(d[i], min(ls->l[i], rs->l[i]));
                r[i] = max(d[i], max(ls->r[i], rs->r[i]));
            }
            return ls->fa = rs->fa = this;
        }
    } pool[M], *null = new P, *pit = pool;
    void upd(P* o, int val) {
        o->val = val;
        for (; o != null; o = o->fa)
            o->Max = max(o->Max, val);
    }
    static P *tmp[M], **pt;
    void init() {
        null->ls = null->rs = null;
        FOR (i, 0, K) null->l[i] = inf, null->r[i] = -inf;
        null->Max = null->val = 0;
    }
    P* build(P** l, P** r, int d = 0) { // [l, r)
        if (d == K) d = 0;
        if (l >= r) return null;
        P** m = l + (r - l) / 2; assert(l <= m && m < r);
        nth_element(l, m, r, [&](const P* a, const P* b){
            return a->d[d] < b->d[d];
        });
        P* o = *m;
        o->ls = build(l, m, d + 1); o->rs = build(m + 1, r, d + 1);
        return o->up();
    }
```

```
    P* Build() {
        pt = tmp; FOR (it, pool, pit) *pt++ = it;
        P* ret = build(tmp, pt); ret->fa = null;
        return ret;
    }
    inline bool inside(int p[], int q[], int l[], int r[]) {
        FOR (i, 0, K) if (r[i] < q[i] || p[i] < l[i]) return false;
        return true;
    }
    int query(P* o, int l[], int r[]) {
        if (o == null) return 0;
        FOR (i, 0, K) if (o->r[i] < l[i] || r[i] < o->l[i]) return 0;
        if (inside(o->l, o->r, l, r)) return o->Max;
        int ret = 0;
        if (o->val > ret && inside(o->d, o->d, l, r)) ret = max(ret, o->val);
        if (o->ls->Max > ret) ret = max(ret, query(o->ls, l, r));
        if (o->rs->Max > ret) ret = max(ret, query(o->rs, l, r));
        return ret;
    }
}
```

- 最近点对
- 要用全局变量大力剪枝

```
namespace kd {
    const int K = 3;
    const int M = N;
    const int inf = 1E9 + 100;
    struct P {
        int d[K];
        int l[K], r[K];
        P *ls, *rs;
        P* up() {
            FOR (i, 0, K) {
                l[i] = min(d[i], min(ls->l[i], rs->l[i]));
                r[i] = max(d[i], max(ls->r[i], rs->r[i]));
            }
            return this;
        }
    } pool[M], *null = new P, *pit = pool;
    static P *tmp[M], **pt;
    void init() {
        null->ls = null->rs = null;
        FOR (i, 0, K) null->l[i] = inf, null->r[i] = -inf;
    }
    P* build(P** l, P** r, int d = 0) { // [l, r)
        if (d == K) d = 0;
        if (l >= r) return null;
        P** m = l + (r - l) / 2;
        nth_element(l, m, r, [&](const P* a, const P* b){
            return a->d[d] < b->d[d];
        });
        P* o = *m;
        o->ls = build(l, m, d + 1); o->rs = build(m + 1, r, d + 1);
        return o->up();
    }
    LL eval(P* o, int d[]) {
        // ...
    }
    LL dist(int d1[], int d2[]) {
        // ...
    }
    LL S;
    LL query(P* o, int d[]) {
        if (o == null) return 0;
        S = max(S, dist(o->d, d));
        LL mdl = eval(o->ls, d), mdr = eval(o->rs, d);
        if (mdl < mdr) {
            if (S > mdl) S = max(S, query(o->ls, d));
            if (S > mdr) S = max(S, query(o->rs, d));
        } else {
```

```
            if (S > mdr) S = max(S, query(o->rs, d));
            if (S > mdl) S = max(S, query(o->ls, d));
        }
        return S;
    }
    P* Build() {
        pt = tmp; FOR (it, pool, pit) *pt++ = it;
        return build(tmp, pt);
    }
}
```

### 树状数组

- 注意：0 是无效下标

```
namespace bit {
    LL c[M];
    inline int lowbit(int x) { return x & -x; }
    void add(int x, LL v) {
        for (int i = x; i < M; i += lowbit(i))
            c[i] += v;
    }
    LL sum(int x) {
        LL ret = 0;
        for (int i = x; i > 0; i -= lowbit(i))
            ret += c[i];
        return ret;
    }
    int kth(LL k) {
        int p = 0;
        for (int lim = 1 << 20; lim; lim /= 2)
            if (p + lim < M && c[p + lim] < k) {
                p += lim;
                k -= c[p];
            }
        return p + 1;
    }
    LL sum(int l, int r) { return sum(r) - sum(l - 1); }
    void add(int l, int r, LL v) { add(l, v); add(r + 1, -v); }
}
```

- 区间修改 & 区间查询（单点修改，查询前缀和的前缀和）

```
namespace bit {
    int c[maxn], cc[maxn];
    inline int lowbit(int x) { return x & -x; }
    void add(int x, int v) {
        for (int i = x; i <= n; i += lowbit(i)) {
            c[i] += v; cc[i] += x * v;
        }
    }
    void add(int l, int r, int v) { add(l, v); add(r + 1, -v); }
    int sum(int x) {
        int ret = 0;
        for (int i = x; i > 0; i -= lowbit(i))
            ret += (x + 1) * c[i] - cc[i];
        return ret;
    }
    int sum(int l, int r) { return sum(r) - sum(l - 1); }
}
```

- 单点修改，查询前缀和的前缀和的前缀和（有用才怪）

```
namespace bit {
    LL c[N], cc[N], ccc[N];
    inline LL lowbit(LL x) { return x & -x; }
    void add(LL x, LL v) {
        for (LL i = x; i < N; i += lowbit(i)) {
            c[i] = (c[i] + v) % MOD;
            cc[i] = (cc[i] + x * v) % MOD;
            ccc[i] = (ccc[i] + x * x % MOD * v) % MOD;
        }
```

```
    }
    void add(LL l, LL r, LL v) { add(l, v); add(r + 1, -v); }
    LL sum(LL x) {
        static LL INV2 = (MOD + 1) / 2;
        LL ret = 0;
        for (LL i = x; i > 0; i -= lowbit(i))
            ret += (x + 1) * (x + 2) % MOD * c[i] % MOD
                    - (2 * x + 3) * cc[i] % MOD
                    + ccc[i];
        return ret % MOD * INV2 % MOD;
    }
    LL sum(LL l, LL r) { return sum(r) - sum(l - 1); }
}
```

- 三维

```
inline int lowbit(int x) { return x & -x; }
void update(int x, int y, int z, int d) {
    for (int i = x; i <= n; i += lowbit(i))
        for (int j = y; j <= n; j += lowbit(j))
            for (int k = z; k <= n; k += lowbit(k))
                c[i][j][k] += d;
}
LL query(int x, int y, int z) {
    LL ret = 0;
    for (int i = x; i > 0; i -= lowbit(i))
        for (int j = y; j > 0; j -= lowbit(j))
            for (int k = z; k > 0; k -= lowbit(k))
                ret += c[i][j][k];
    return ret;
}
LL solve(int x, int y, int z, int xx, int yy, int zz) {
    return    query(xx, yy, zz)
            - query(xx, yy, z - 1)
            - query(xx, y - 1, zz)
            - query(x - 1, yy, zz)
            + query(xx, y - 1, z - 1)
            + query(x - 1, yy, z - 1)
            + query(x - 1, y - 1, zz)
            - query(x - 1, y - 1, z - 1);
```

## 主席树

- 正常主席树

```
namespace tree {
#define mid ((l + r) >> 1)
#define lson l, mid
#define rson mid + 1, r
    const int MAGIC = M * 30;
    struct P {
        int sum, ls, rs;
    } tr[MAGIC] = {{0, 0, 0}};
    int sz = 1;
    int N(int sum, int ls, int rs) {
        if (sz == MAGIC) assert(0);
        tr[sz] = {sum, ls, rs};
        return sz++;
    }
    int ins(int o, int x, int v, int l = 1, int r = ls) {
        if (x < l || x > r) return o;
        const P& t = tr[o];
        if (l == r) return N(t.sum + v, 0, 0);
        return N(t.sum + v, ins(t.ls, x, v, lson), ins(t.rs, x, v, rson));
    }
    int query(int o, int ql, int qr, int l = 1, int r = ls) {
        if (ql > r || l > qr) return 0;
        const P& t = tr[o];
        if (ql <= l && r <= qr) return t.sum;
        return query(t.ls, ql, qr, lson) + query(t.rs, ql, qr, rson);
    }
```

```
}
```

- 第 k 大

```
struct TREE {
#define mid ((l + r) >> 1)
#define lson l, mid
#define rson mid + 1, r
    struct P {
        int w, ls, rs;
    } tr[maxn * 20];
    int sz = 1;
    TREE() { tr[0] = {0, 0, 0}; }
    int N(int w, int ls, int rs) {
        tr[sz] = {w, ls, rs};
        return sz++;
    }
    int ins(int tt, int l, int r, int x) {
        if (x < l || r < x) return tt;
        const P& t = tr[tt];
        if (l == r) return N(t.w + 1, 0, 0);
        return N(t.w + 1, ins(t.ls, lson, x), ins(t.rs, rson, x));
    }
    int query(int pp, int qq, int l, int r, int k) { // (pp, qq]
        if (l == r) return l;
        const P &p = tr[pp], &q = tr[qq];
        int w = tr[q.ls].w - tr[p.ls].w;
        if (k <= w) return query(p.ls, q.ls, lson, k);
        else return query(p.rs, q.rs, rson, k - w);
    }
} tree;
```

- 树状数组套主席树

```
typedef vector<int> VI;
struct TREE {
#define mid ((l + r) >> 1)
#define lson l, mid
#define rson mid + 1, r
    struct P {
        int w, ls, rs;
    } tr[maxn * 20 * 20];
    int sz = 1;
    TREE() { tr[0] = {0, 0, 0}; }
    int N(int w, int ls, int rs) {
        tr[sz] = {w, ls, rs};
        return sz++;
    }
    int add(int tt, int l, int r, int x, int d) {
        if (x < l || r < x) return tt;
        const P& t = tr[tt];
        if (l == r) return N(t.w + d, 0, 0);
        return N(t.w + d, add(t.ls, lson, x, d), add(t.rs, rson, x, d));
    }
    int ls_sum(const VI& rt) {
        int ret = 0;
        FOR (i, 0, rt.size())
            ret += tr[tr[rt[i]].ls].w;
        return ret;
    }
    inline void ls(VI& rt) { transform(rt.begin(), rt.end(), rt.begin(), [&](int x)->int{ return tr[x].ls; }); }
    inline void rs(VI& rt) { transform(rt.begin(), rt.end(), rt.begin(), [&](int x)->int{ return tr[x].rs; }); }
    int query(VI& p, VI& q, int l, int r, int k) {
        if (l == r) return l;
        int w = ls_sum(q) - ls_sum(p);
        if (k <= w) {
            ls(p); ls(q);
            return query(p, q, lson, k);
        }
        else {
            rs(p); rs(q);
            return query(p, q, rson, k - w);
```

```
        }
    }
} tree;
struct BIT {
    int root[maxn];
    void init() { memset(root, 0, sizeof root); }
    inline int lowbit(int x) { return x & -x; }
    void update(int p, int x, int d) {
        for (int i = p; i <= m; i += lowbit(i))
            root[i] = tree.add(root[i], 1, m, x, d);
    }
    int query(int l, int r, int k) {
        VI p, q;
        for (int i = l - 1; i > 0; i -= lowbit(i)) p.push_back(root[i]);
        for (int i = r; i > 0; i -= lowbit(i)) q.push_back(root[i]);
        return tree.query(p, q, 1, m, k);
    }
} bit;

void init() {
    m = 10000;
    tree.sz = 1;
    bit.init();
    FOR (i, 1, m + 1)
        bit.update(i, a[i], 1);
}
```

## 左偏树

```
namespace LTree {
    extern struct P* null, *pit;
    queue<P*> trash;
    const int M = 1E5 + 100;
    struct P {
        P *ls, *rs;
        LL v;
        int d;
        void operator delete (void* ptr) {
            trash.push((P*)ptr);
        }
        void* operator new(size_t size) {
            if (trash.empty()) return pit++;
            void* ret = trash.front(); trash.pop(); return ret;
        }

        void prt() {
            if (this == null) return;
            cout << v << ' ';
            ls->prt(); rs->prt();
        }
    } pool[M], *pit = pool, *null = new P{0, 0, -1, -1};
    P* N(LL v) {
        return new P{null, null, v, 0};
    }
    P* merge(P* a, P* b) {
        if (a == null) return b;
        if (b == null) return a;
        if (a->v > b->v) swap(a, b);
        a->rs = merge(a->rs, b);
        if (a->ls->d < a->rs->d) swap(a->ls, a->rs);
        a->d = a->rs->d + 1;
        return a;
    }

    LL pop(P*& o) {
        LL ret = o->v;
        P* t = o;
        o = merge(o->ls, o->rs);
        delete t;
        return ret;
    }
```

```
}
```

可持久化

```
namespace LTree {
    extern struct P* null, *pit;
    queue<P*> trash;
    const int M = 1E6 + 100;
    struct P {
        P *ls, *rs;
        LL v;
        int d;
        void operator delete (void* ptr) {
            trash.push((P*)ptr);
        }
        void* operator new(size_t size) {
            if (trash.empty()) return pit++;
            void* ret = trash.front(); trash.pop(); return ret;
        }
    } pool[M], *pit = pool, *null = new P{0, 0, -1, -1};
    P* N(LL v, P* ls = null, P* rs = null) {
        if (ls->d < rs->d) swap(ls, rs);
        return new P{ls, rs, v, rs->d + 1};
    }
    P* merge(P* a, P* b) {
        if (a == null) return b;
        if (b == null) return a;
        if (a->v < b->v)
            return N(a->v, a->ls, merge(a->rs, b));
        else
            return N(b->v, b->ls, merge(b->rs, a));
    }

    LL pop(P*& o) {
        LL ret = o->v;
        o = merge(o->ls, o->rs);
        return ret;
    }
}
```

## Treap

- 非旋 Treap
- v 小根堆
- 模板题 bzoj 3224
- lower 第一个大于等于的是第几个 (0-based)
- upper 第一个大于的是第几个 (0-based)
- split 左侧分割出 rk 个元素
- 树套树略
- hkk 版

```
#define lc c[0]
#define rc c[1]
struct Node { int c[2], v, s, r; } t[N]; int nod;
int newnode(int x) { t[++nod].v = x, t[nod].s = 1, t[nod].r = rand(); return nod; }
void pushup(int o) { t[o].s = t[t[o].lc].s + t[t[o].rc].s + 1; }
void split(int o, int v, int &x, int &y) {
    if (!o) return x = y = 0, (void)0;
    if (t[o].v <= v) x = o, split(t[x].rc, v, t[x].rc, y), pushup(x);
    else y = o, split(t[y].lc, v, x, t[y].lc), pushup(y);
}
int merge(int x, int y) {
    if (!x || !y) return x ^ y;
    if (t[x].r < t[y].r) return t[x].rc = merge(t[x].rc, y), pushup(x), x;
    else return t[y].lc = merge(x, t[y].lc), pushup(y), y;
}
void qins(int v) {
    int x, y;
    split(rt, v, x, y);
```

```cpp
    rt = merge(merge(x, newnode(v)), y);
}
void qdel(int v) {
    int x, y, z;
    split(rt, v, x, y);
    split(x, v - 1, x, z);
    rt = merge(merge(x, merge(t[z].lc, t[z].rc)), y);
}
int qrank(int v) {
    int ans = 0, o = rt;
    while (o)
        if (v <= t[o].v) o = t[o].lc;
        else ans += t[t[o].lc].s + 1, o = t[o].rc;
    return ans + 1;
}
int qkth(int k) {
    int o = rt;
    while (o) {
        if (t[t[o].lc].s + 1 == k) return t[o].v;
        if (k <= t[t[o].lc].s) o = t[o].lc;
        else k -= t[t[o].lc].s + 1, o = t[o].rc;
    }
}
int qpre(int v) {
    int ans = -INF, o = rt;
    while (o)
        if (v <= t[o].v) o = t[o].lc;
        else smax(ans, t[o].v), o = t[o].rc;
    return ans;
}
int qnxt(int v) {
    int ans = INF, o = rt;
    while (o)
        if (v >= t[o].v) o = t[o].rc;
        else smin(ans, t[o].v), o = t[o].lc;
    return ans;
}
```

- 模板原版

```cpp
namespace treap {
    const int M = maxn * 17;
    extern struct P* const null;
    struct P {
        P *ls, *rs;
        int v, sz;
        unsigned rd;
        P(int v): ls(null), rs(null), v(v), sz(1), rd(rnd()) {}
        P(): sz(0) {}

        P* up() { sz = ls->sz + rs->sz + 1; return this; }
        int lower(int v) {
            if (this == null) return 0;
            return this->v >= v ? ls->lower(v) : rs->lower(v) + ls->sz + 1;
        }
        int upper(int v) {
            if (this == null) return 0;
            return this->v > v ? ls->upper(v) : rs->upper(v) + ls->sz + 1;
        }
    } *const null = new P, pool[M], *pit = pool;

    P* merge(P* l, P* r) {
        if (l == null) return r; if (r == null) return l;
        if (l->rd < r->rd) { l->rs = merge(l->rs, r); return l->up(); }
        else { r->ls = merge(l, r->ls); return r->up(); }
    }

    void split(P* o, int rk, P*& l, P*& r) {
        if (o == null) { l = r = null; return; }
        if (o->ls->sz >= rk) { split(o->ls, rk, l, o->ls); r = o->up(); }
        else { split(o->rs, rk - o->ls->sz - 1, o->rs, r); l = o->up(); }
    }
```

```
}
```

- 持久化 Treap

```
namespace treap {
    const int M = maxn * 17 * 12;
    extern struct P* const null, *pit;
    struct P {
        P *ls, *rs;
        int v, sz;
        LL sum;
        P(P* ls, P* rs, int v): ls(ls), rs(rs), v(v), sz(ls->sz + rs->sz + 1),
                                                    sum(ls->sum + rs->sum + v) {}
        P() {}

        void* operator new(size_t _) { return pit++; }
        template<typename T>
        int rk(int v, T&& cmp) {
            if (this == null) return 0;
            return cmp(this->v, v) ? ls->rk(v, cmp) : rs->rk(v, cmp) + ls->sz + 1;
        }
        int lower(int v) { return rk(v, greater_equal<int>()); }
        int upper(int v) { return rk(v, greater<int>()); }
    } pool[M], *pit = pool, *const null = new P;
    P* merge(P* l, P* r) {
        if (l == null) return r; if (r == null) return l;
        if (rnd() % (l->sz + r->sz) < l->sz) return new P{l->ls, merge(l->rs, r), l->v};
        else return new P{merge(l, r->ls), r->rs, r->v};
    }
    void split(P* o, int rk, P*& l, P*& r) {
        if (o == null) { l = r = null; return; }
        if (o->ls->sz >= rk) { split(o->ls, rk, l, r); r = new P{r, o->rs, o->v}; }
        else { split(o->rs, rk - o->ls->sz - 1, l, r); l = new P{o->ls, l, o->v}; }
    }
}
```

- 带 pushdown 的持久化 Treap
- 注意任何修改操作前一定要 FIX

```
int now;
namespace Treap {
    const int M = 10000000;
    extern struct P* const null, *pit;
    struct P {
        P *ls, *rs;
        int sz, time;
        LL cnt, sc, pos, add;
        bool rev;

        P* up() { sz = ls->sz + rs->sz + 1; sc = ls->sc + rs->sc + cnt; return this; } // MOD
        P* check() {
            if (time == now) return this;
            P* t = new(pit++) P; *t = *this; t->time = now; return t;
        };
        P* _do_rev() { rev ^= 1; add *= -1; pos *= -1; swap(ls, rs); return this; } // MOD
        P* _do_add(LL v) { add += v; pos += v; return this; } // MOD
        P* do_rev() { if (this == null) return this; return check()->_do_rev(); } // FIX & MOD
        P* do_add(LL v) { if (this == null) return this; return check()->_do_add(v); } // FIX & MOD
        P* _down() { // MOD
            if (rev) { ls = ls->do_rev(); rs = rs->do_rev(); rev = 0; }
            if (add) { ls = ls->do_add(add); rs = rs->do_add(add); add = 0; }
            return this;
        }
        P* down() { return check()->_down(); } // FIX & MOD
        void _split(LL p, P*& l, P*& r) { // MOD
            if (pos >= p) { ls->split(p, l, r); ls = r; r = up(); }
            else          { rs->split(p, l, r); rs = l; l = up(); }
        }
        void split(LL p, P*& l, P*& r) { // FIX & MOD
            if (this == null) l = r = null;
            else down()->_split(p, l, r);
        }
```

```
    } pool[M], *pit = pool, *const null = new P;
    P* merge(P* a, P* b) {
        if (a == null) return b; if (b == null) return a;
        if (rand() % (a->sz + b->sz) < a->sz) { a = a->down(); a->rs = merge(a->rs, b); return a->up(); }
        else                                  { b = b->down(); b->ls = merge(a, b->ls); return b->up(); }
    }
}
```

## Treap-序列

- 区间 ADD，SUM

```
namespace treap {
    const int M = 8E5 + 100;
    extern struct P*const null;
    struct P {
        P *ls, *rs;
        int sz, val, add, sum;
        P(int v, P* ls = null, P* rs = null): ls(ls), rs(rs), sz(1), val(v), add(0), sum(v) {}
        P(): sz(0), val(0), add(0), sum(0) {}

        P* up() {
            assert(this != null);
            sz = ls->sz + rs->sz + 1;
            sum = ls->sum + rs->sum + val + add * sz;
            return this;
        }
        void upd(int v) {
            if (this == null) return;
            add += v;
            sum += sz * v;
        }
        P* down() {
            if (add) {
                ls->upd(add); rs->upd(add);
                val += add;
                add = 0;
            }
            return this;
        }

        P* select(int rk) {
            if (rk == ls->sz + 1) return this;
            return ls->sz >= rk ? ls->select(rk) : rs->select(rk - ls->sz - 1);
        }
    } pool[M], *pit = pool, *const null = new P, *rt = null;

    P* merge(P* a, P* b) {
        if (a == null) return b->up();
        if (b == null) return a->up();
        if (rand() % (a->sz + b->sz) < a->sz) {
            a->down()->rs = merge(a->rs, b);
            return a->up();
        } else {
            b->down()->ls = merge(a, b->ls);
            return b->up();
        }
    }

    void split(P* o, int rk, P*& l, P*& r) {
        if (o == null) { l = r = null; return; }
        o->down();
        if (o->ls->sz >= rk) {
            split(o->ls, rk, l, o->ls);
            r = o->up();
        } else {
            split(o->rs, rk - o->ls->sz - 1, o->rs, r);
            l = o->up();
        }
    }

```

```
    inline void insert(int k, int v) {
        P *l, *r;
        split(rt, k - 1, l, r);
        rt = merge(merge(l, new (pit++) P(v)), r);
    }

    inline void erase(int k) {
        P *l, *r, *_, *t;
        split(rt, k - 1, l, t);
        split(t, 1, _, r);
        rt = merge(l, r);
    }

    P* build(int l, int r, int* a) {
        if (l > r) return null;
        if (l == r) return new(pit++) P(a[l]);
        int m = (l + r) / 2;
        return (new(pit++) P(a[m], build(l, m - 1, a), build(m + 1, r, a)))->up();
    }
};
```

- 区间 REVERSE，ADD，MIN

```
namespace treap {
    extern struct P*const null;
    struct P {
        P *ls, *rs;
        int sz, v, add, m;
        bool flip;
        P(int v, P* ls = null, P* rs = null): ls(ls), rs(rs), sz(1), v(v), add(0), m(v), flip(0) {}
        P(): sz(0), v(INF), m(INF) {}

        void upd(int v) {
            if (this == null) return;
            add += v; m += v;
        }
        void rev() {
            if (this == null) return;
            swap(ls, rs);
            flip ^= 1;
        }
        P* up() {
            assert(this != null);
            sz = ls->sz + rs->sz + 1;
            m = min(min(ls->m, rs->m), v) + add;
            return this;
        }
        P* down() {
            if (add) {
                ls->upd(add); rs->upd(add);
                v += add;
                add = 0;
            }
            if (flip) {
                ls->rev(); rs->rev();
                flip = 0;
            }
            return this;
        }

        P* select(int k) {
            if (ls->sz + 1 == k) return this;
            if (ls->sz >= k) return ls->select(k);
            return rs->select(k - ls->sz - 1);
        }

    } pool[M], *const null = new P, *pit = pool, *rt = null;

    P* merge(P* a, P* b) {
        if (a == null) return b;
        if (b == null) return a;
        if (rnd() % (a->sz + b->sz) < a->sz) {
```

```cpp
            a->down()->rs = merge(a->rs, b);
            return a->up();
        } else {
            b->down()->ls = merge(a, b->ls);
            return b->up();
        }
    }

    void split(P* o, int k, P*& l, P*& r) {
        if (o == null) { l = r = null; return; }
        o->down();
        if (o->ls->sz >= k) {
            split(o->ls, k, l, o->ls);
            r = o->up();
        } else {
            split(o->rs, k - o->ls->sz - 1, o->rs, r);
            l = o->up();
        }
    }

    P* build(int l, int r, int* v) {
        if (l > r) return null;
        int m = (l + r) >> 1;
        return (new (pit++) P(v[m], build(l, m - 1, v), build(m + 1, r, v)))->up();
    }

    void go(int x, int y, void f(P*&)) {
        P *l, *m, *r;
        split(rt, y, l, r);
        split(l, x - 1, l, m);
        f(m);
        rt = merge(merge(l, m), r);
    }
}
using namespace treap;
int a[maxn], n, x, y, Q, v, k, d;
char s[100];

int main() {
    cin >> n;
    FOR (i, 1, n + 1) scanf("%d", &a[i]);
    rt = build(1, n, a);
    cin >> Q;
    while (Q--) {
        scanf("%s", s);
        if (s[0] == 'A') {
            scanf("%d%d%d", &x, &y, &v);
            go(x, y, [](P*& o){ o->upd(v); });
        } else if (s[0] == 'R' && s[3] == 'E') {
            scanf("%d%d", &x, &y);
            go(x, y, [](P*& o){ o->rev(); });
        } else if (s[0] == 'R' && s[3] == 'O') {
            scanf("%d%d%d", &x, &y, &d);
            d %= y - x + 1;
            go(x, y, [](P*& o){
                P *l, *r;
                split(o, o->sz - d, l, r);
                o = merge(r, l);
            });
        } else if (s[0] == 'I') {
            scanf("%d%d", &k, &v);
            go(k + 1, k, [](P*& o){ o = new (pit++) P(v); });
        } else if (s[0] == 'D') {
            scanf("%d", &k);
            go(k, k, [](P*& o){ o = null; });
        } else if (s[0] == 'M') {
            scanf("%d%d", &x, &y);
            go(x, y, [](P*& o) {
                printf("%d\n", o->m);
            });
        }
```

```
    }
}
```

- 持久化

```
namespace treap {
    struct P;
    extern P*const null;
    P* N(P* ls, P* rs, LL v, bool fill);
    struct P {
        P *const ls, *const rs;
        const int sz, v;
        const LL sum;
        bool fill;
        int cnt;

        void split(int k, P*& l, P*& r) {
            if (this == null) { l = r = null; return; }
            if (ls->sz >= k) {
                ls->split(k, l, r);
                r = N(r, rs, v, fill);
            } else {
                rs->split(k - ls->sz - fill, l, r);
                l = N(ls, l, v, fill);
            }
        }


    } *const null = new P{0, 0, 0, 0, 0, 0, 1};

    P* N(P* ls, P* rs, LL v, bool fill) {
        ls->cnt++; rs->cnt++;
        return new P{ls, rs, ls->sz + rs->sz + fill, v, ls->sum + rs->sum + v, fill, 1};
    }

    P* merge(P* a, P* b) {
        if (a == null) return b;
        if (b == null) return a;
        if (rand() % (a->sz + b->sz) < a->sz)
            return N(a->ls, merge(a->rs, b), a->v, a->fill);
        else
            return N(merge(a, b->ls), b->rs, b->v, b->fill);
    }

    void go(P* o, int x, int y, P*& l, P*& m, P*& r) {
        o->split(y, l, r);
        l->split(x - 1, l, m);
    }
}
```

## 可回滚并查集

- 注意这个不是可持久化并查集
- 查找时不进行路径压缩
- 复杂度靠按秩合并解决

```
namespace uf {
    int fa[maxn], sz[maxn];
    int undo[maxn], top;
    void init() { memset(fa, -1, sizeof fa); memset(sz, 0, sizeof sz); top = 0; }
    int findset(int x) { while (fa[x] != -1) x = fa[x]; return x; }
    bool join(int x, int y) {
        x = findset(x); y = findset(y);
        if (x == y) return false;
        if (sz[x] > sz[y]) swap(x, y);
        undo[top++] = x;
        fa[x] = y;
        sz[y] += sz[x] + 1;
        return true;
    }
    inline int checkpoint() { return top; }
```

```
    void rewind(int t) {
        while (top > t) {
            int x = undo[--top];
            sz[fa[x]] -= sz[x] + 1;
            fa[x] = -1;
        }
    }
}
```

### 舞蹈链

- 注意 link 的 y 的范围是 [1, n]
- 注意在某些情况下替换掉 memset
- 精确覆盖

```
struct P {
    P *L, *R, *U, *D;
    int x, y;
};

const int INF = 1E9;

struct DLX {
#define TR(i, D, s) for (P* i = s->D; i != s; i = i->D)
    static const int M = 2E5;
    P pool[M], *h[M], *r[M], *pit;
    int sz[M];
    bool solved;
    stack<int> ans;
    void init(int n) {
        pit = pool;
        ++n;
        solved = false;
        while (!ans.empty()) ans.pop();
        memset(r, 0, sizeof r);
        memset(sz, 0, sizeof sz);
        FOR (i, 0, n)
            h[i] = new (pit++) P;
        FOR (i, 0, n) {
            h[i]->L = h[(i + n - 1) % n];
            h[i]->R = h[(i + 1) % n];
            h[i]->U = h[i]->D = h[i];
            h[i]->y = i;
        }
    }

    void link(int x, int y) {
        sz[y]++;
        auto p = new (pit++) P;
        p->x = x; p->y = y;
        p->U = h[y]->U; p->D = h[y];
        p->D->U = p->U->D = p;
        if (!r[x]) r[x] = p->L = p->R = p;
        else {
            p->L = r[x]; p->R = r[x]->R;
            p->L->R = p->R->L = p;
        }
    }

    void remove(P* p) {
        p->L->R = p->R; p->R->L = p->L;
        TR (i, D, p)
            TR (j, R, i) {
                j->D->U = j->U; j->U->D = j->D;
                sz[j->y]--;
            }
    }

    void recall(P* p) {
```

```
        p->L->R = p->R->L = p;
        TR (i, U, p)
            TR (j, L, i) {
                j->D->U = j->U->D = j;
                sz[j->y]++;
            }
    }

    bool dfs(int d) {
        if (solved) return true;
        if (h[0]->R == h[0]) return solved = true;
        int m = INF;
        P* c;
        TR (i, R, h[0])
            if (sz[i->y] < m) { m = sz[i->y]; c = i; }
        remove(c);
        TR (i, D, c) {
            ans.push(i->x);
            TR (j, R, i) remove(h[j->y]);
            if (dfs(d + 1)) return true;
            TR (j, L, i) recall(h[j->y]);
            ans.pop();
        }
        recall(c);
        return false;
    }
} dlx;
```

- 可重复覆盖

```
struct P {
    P *L, *R, *U, *D;
    int x, y;
};

const int INF = 1E9;

struct DLX {
#define TR(i, D, s) for (P* i = s->D; i != s; i = i->D)
    static const int M = 2E5;
    P pool[M], *h[M], *r[M], *pit;
    int sz[M], vis[M], ans, clk;
    void init(int n) {
        clk = 0;
        ans = INF;
        pit = pool;
        ++n;
        memset(r, 0, sizeof r);
        memset(sz, 0, sizeof sz);
        memset(vis, -1, sizeof vis);
        FOR (i, 0, n)
            h[i] = new (pit++) P;
        FOR (i, 0, n) {
            h[i]->L = h[(i + n - 1) % n];
            h[i]->R = h[(i + 1) % n];
            h[i]->U = h[i]->D = h[i];
            h[i]->y = i;
        }
    }

    void link(int x, int y) {
        sz[y]++;
        auto p = new (pit++) P;
        p->x = x; p->y = y;
        p->U = h[y]->U; p->D = h[y];
        p->D->U = p->U->D = p;
        if (!r[x]) r[x] = p->L = p->R = p;
        else {
            p->L = r[x]; p->R = r[x]->R;
            p->L->R = p->R->L = p;
        }
    }
```

```

    void remove(P* p) {
        TR (i, D, p) {
            i->L->R = i->R;
            i->R->L = i->L;
        }
    }

    void recall(P* p) {
        TR (i, U, p)
            i->L->R = i->R->L = i;
    }

    int eval() {
        ++clk;
        int ret = 0;
        TR (i, R, h[0])
            if (vis[i->y] != clk) {
                ++ret;
                vis[i->y] = clk;
                TR (j, D, i)
                    TR (k, R, j)
                        vis[k->y] = clk;
            }
        return ret;
    }

    void dfs(int d) {
        if (h[0]->R == h[0]) { ans = min(ans, d); return; }
        if (eval() + d >= ans) return;
        P* c;
        int m = INF;
        TR (i, R, h[0])
            if (sz[i->y] < m) { m = sz[i->y]; c = i; }
        TR (i, D, c) {
            remove(i);
            TR (j, R, i) remove(j);
            dfs(d + 1);
            TR (j, L, i) recall(j);
            recall(i);
        }
    }
} dlx;
```

## CDQ 分治

```
const int maxn = 2E5 + 100;
struct P {
    int x, y;
    int* f;
    bool d1, d2;
} a[maxn], b[maxn], c[maxn];
int f[maxn];

void go2(int l, int r) {
    if (l + 1 == r) return;
    int m = (l + r) >> 1;
    go2(l, m); go2(m, r);
    FOR (i, l, m) b[i].d2 = 0;
    FOR (i, m, r) b[i].d2 = 1;
    merge(b + l, b + m, b + m, b + r, c + l, [](const P& a, const P& b)->bool {
            if (a.y != b.y) return a.y < b.y;
            return a.d2 > b.d2;
        });
    int mx = -1;
    FOR (i, l, r) {
        if (c[i].d1 && c[i].d2) *c[i].f = max(*c[i].f, mx + 1);
        if (!c[i].d1 && !c[i].d2) mx = max(mx, *c[i].f);
    }
    FOR (i, l, r) b[i] = c[i];
}
```

```
void go1(int l, int r) { // [l, r)
    if (l + 1 == r) return;
    int m = (l + r) >> 1;
    go1(l, m);
    FOR (i, l, m) a[i].d1 = 0;
    FOR (i, m, r) a[i].d1 = 1;
    copy(a + l, a + r, b + l);
    sort(b + l, b + r, [](const P& a, const P& b)->bool {
            if (a.x != b.x) return a.x < b.x;
            return a.d1 > b.d1;
        });
    go2(l, r);
    go1(m, r);
}
```

- k 维 LIS

```
struct P {
    int v[K];
    LL f;
    bool d[K];
} o[N << 10];
P* a[K][N << 10];
int k;
void go(int now, int l, int r) {
    if (now == 0) {
        if (l + 1 == r) return;
        int m = (l + r) / 2;
        go(now, l, m);
        FOR (i, l, m) a[now][i]->d[now] = 0;
        FOR (i, m, r) a[now][i]->d[now] = 1;
        copy(a[now] + l, a[now] + r, a[now + 1] + l);
        sort(a[now + 1] + l, a[now + 1] + r, [now](const P* a, const P* b){
            if (a->v[now] != b->v[now]) return a->v[now] < b->v[now];
            return a->d[now] > b->d[now];
        });
        go(now + 1, l, r);
        go(now, m, r);
    } else {
        if (l + 1 == r) return;
        int m = (l + r) / 2;
        go(now, l, m); go(now, m, r);
        FOR (i, l, m) a[now][i]->d[now] = 0;
        FOR (i, m, r) a[now][i]->d[now] = 1;
        merge(a[now] + l, a[now] + m, a[now] + m, a[now] + r, a[now + 1] + l, [now](const P* a, const P* b){
            if (a->v[now] != b->v[now]) return a->v[now] < b->v[now];
            return a->d[now] > b->d[now];
        });
        copy(a[now + 1] + l, a[now + 1] + r, a[now] + l);
        if (now < k - 2) {
            go(now + 1, l, r);
        } else {
            LL sum = 0;
            FOR (i, l, r) {
                dbg(a[now][i]->v[0], a[now][i]->v[1], a[now][i]->f,
                                  a[now][i]->d[0], a[now][i]->d[1]);
                int cnt = 0;
                FOR (j, 0, now + 1) cnt += a[now][i]->d[j];
                if (cnt == 0) {
                    sum += a[now][i]->f;
                } else if (cnt == now + 1) {
                    a[now][i]->f = (a[now][i]->f + sum) % MOD;
                }
            }
        }
    }
}
```

### 笛卡尔树

```
void build(const vector<int>& a) {
    static P *stack[M], *x, *last;
    int p = 0;
    FOR (i, 0, a.size()) {
        x = new P(i + 1, a[i]);
        last = null;
        while (p && stack[p - 1]->v > x->v) {
            stack[p - 1]->maintain();
            last = stack[--p];
        }
        if (p) stack[p - 1]->rs = x;
        x->ls = last;
        stack[p++] = x;
    }
    while (p)
        stack[--p]->maintain();
    rt = stack[0];
}
```

```
void build() {
    static int s[N], last;
    int p = 0;
    FOR (x, 1, n + 1) {
        last = 0;
        while (p && val[s[p - 1]] > val[x]) last = s[--p];
        if (p) G[s[p - 1]][1] = x;
        if (last) G[x][0] = last;
        s[p++] = x;
    }
    rt = s[0];
}
```

### Trie

- 二进制 Trie

```
namespace trie {
    const int M = 31;
    int ch[N * M][2], sz;
    void init() { memset(ch, 0, sizeof ch); sz = 2; }
    void ins(LL x) {
        int u = 1;
        FORD (i, M, -1) {
            bool b = x & (1LL << i);
            if (!ch[u][b]) ch[u][b] = sz++;
            u = ch[u][b];
        }
    }
}
```

- 持久化二进制 Trie
- sz=1

```
struct P { int w, ls, rs; };
P tr[M] = {{0, 0, 0}};
int sz;

int _new(int w, int ls, int rs) { tr[sz] = {w, ls, rs}; return sz++; }
int ins(int oo, int v, int d = 30) {
    P& o = tr[oo];
    if (d == -1) return _new(o.w + 1, 0, 0);
    bool u = v & (1 << d);
    return _new(o.w + 1, u == 0 ? ins(o.ls, v, d - 1) : o.ls, u == 1 ? ins(o.rs, v, d - 1) : o.rs);
}
int query(int pp, int qq, int v, int d = 30) {
    if (d == -1) return 0;
    bool u = v & (1 << d);
    P &p = tr[pp], &q = tr[qq];
    int lw = tr[q.ls].w - tr[p.ls].w;
    int rw = tr[q.rs].w - tr[p.rs].w;
```

```cpp

    int ret = 0;
    if (u == 0) {
        if (rw) { ret += 1 << d; ret += query(p.rs, q.rs, v, d - 1); }
        else ret += query(p.ls, q.ls, v, d - 1);
    } else {
        if (lw) { ret += 1 << d; ret += query(p.ls, q.ls, v, d - 1); }
        else ret += query(p.rs, q.rs, v, d - 1);
    }
    return ret;
}
```

## exSTL

### 优先队列

- binary_heap_tag
- pairing_heap_tag 支持修改
- thin_heap_tag 如果修改只有 increase 则较快，不支持 join

```cpp
#include<ext/pb_ds/priority_queue.hpp>
using namespace __gnu_pbds;

typedef __gnu_pbds::priority_queue<LL, less<LL>, pairing_heap_tag> PQ;
__gnu_pbds::priority_queue<int, cmp, pairing_heap_tag>::point_iterator it;
PQ pq, pq2;

int main() {
    auto it = pq.push(2);
    pq.push(3);
    assert(pq.top() == 3);
    pq.modify(it, 4);
    assert(pq.top() == 4);
    pq2.push(5);
    pq.join(pq2);
    assert(pq.top() == 5);
}
```

### 平衡树

- ov_tree_tag
- rb_tree_tag
- splay_tree_tag
- mapped: null_type 或 null_mapped_type（旧版本）为空
- Node_Update 为 tree_order_statistics_node_update 时才可以 find_by_order & order_of_key
- find_by_order 找 order + 1 小的元素（其实都是从 0 开始计数），或者有 order 个元素比它小的 key
- order_of_key 有多少个比 r_key 小的元素
- join & split

```cpp
#include <ext/pb_ds/assoc_container.hpp>
using namespace __gnu_pbds;
using Tree = tree<int, null_type, less<int>, rb_tree_tag, tree_order_statistics_node_update>;
Tree t;
```

### 持久化平衡树

```cpp
#include <ext/rope>
using namespace __gnu_cxx;
rope<int> s;

int main() {
    FOR (i, 0, 5) s.push_back(i); // 0 1 2 3 4
    s.replace(1, 2, s); // 0 (0 1 2 3 4) 3 4
```

```cpp
    auto ss = s.substr(2, 2); // 1 2、
    s.erase(2, 2); // 0 1 4
    s.insert(2, s); // equal to s.replace(2, 0, s)
    assert(s[2] == s.at(2)); // 2
}
```

**哈希表**

```cpp
#include<ext/pb_ds/assoc_container.hpp>
#include<ext/pb_ds/hash_policy.hpp>
using namespace __gnu_pbds;

gp_hash_table<int, int> mp;
cc_hash_table<int, int> mp;
```

## Link-Cut Tree

- 图中相邻的结点在伸展树中不一定是父子关系
- 遇事不决 `make_root`
- 跑左右儿子的时候不要忘记 down
- hkk 版

```cpp
#define lc c[0]
#define rc c[1]
struct Node { int c[2], fa, v, sum; bool rev; } t[N];
int S[N];
bool isroot(int o) { return t[t[o].fa].lc != o && t[t[o].fa].rc != o;}
bool idtfy(int o) { return t[t[o].fa].rc == o; }
void connect(int fa, int o, bool d) { t[fa].c[d] = o, t[o].fa = fa; }
void pushup(int o) { t[o].sum = t[t[o].lc].sum ^ t[t[o].rc].sum ^ t[o].v; }
void pushdown(int o) {
    if (!t[o].rev) return;
    t[o].rev = 0;
    if (t[o].lc) std::swap(t[t[o].lc].lc, t[t[o].lc].rc), t[t[o].lc].rev ^= 1;
    if (t[o].rc) std::swap(t[t[o].rc].lc, t[t[o].rc].rc), t[t[o].rc].rev ^= 1;
}
void rotate(int o) {
    int fa = t[o].fa, pa = t[fa].fa, d1 = idtfy(o), d2 = idtfy(fa), b = t[o].c[d1 ^ 1];
    t[o].fa = pa, !isroot(fa) && (t[pa].c[d2] = o);
    connect(fa, b, d1), connect(o, fa, d1 ^ 1);
    pushup(fa), pushup(o);
}
void splay(int o) {
    int x = o, tp = 1;
    S[tp] = x;
    while (!isroot(x)) S[++tp] = x = t[x].fa;
    while (tp) pushdown(S[tp--]);
    while (!isroot(o)) {
        int fa = t[o].fa;
        if (isroot(fa)) rotate(o);
        else if (idtfy(o) == idtfy(fa)) rotate(fa), rotate(o);
        else rotate(o), rotate(o);
    }
}
void access(int o) {
    for (int x = 0; o; x = o, o = t[o].fa)
        splay(o), t[o].rc = x, pushup(o);
}
int findrt(int x) {
    access(x), splay(x);
    while (t[x].lc) pushdown(x), x = t[x].lc;
    return splay(x), x;
}
void mkrt(int x) {
    access(x), splay(x);
    t[x].rev ^= 1, std::swap(t[x].lc, t[x].rc);
}
void split(int x, int y) { mkrt(x), access(y), splay(y); }
void link(int x, int y) {
    mkrt(x);
```

```cpp
    if (findrt(y) != x) t[x].fa = y;
}
void cut(int x, int y) {
    split(x, y);
    if (t[y].lc == x && t[x].fa == y) t[y].lc = t[x].fa = 0, pushup(y);
}
```

- 模板原版

```cpp
namespace lct {
    extern struct P *const null;
    const int M = N;
    struct P {
        P *fa, *ls, *rs;
        int v, maxv;
        bool rev;

        bool has_fa() { return fa->ls == this || fa->rs == this; }
        bool d() { return fa->ls == this; }
        P*& c(bool x) { return x ? ls : rs; }
        void do_rev() {
            if (this == null) return;
            rev ^= 1;
            swap(ls, rs);
        }
        P* up() {
            maxv = max(v, max(ls->maxv, rs->maxv));
            return this;
        }
        void down() {
            if (rev) {
                rev = 0;
                ls->do_rev(); rs->do_rev();
            }
        }
        void all_down() { if (has_fa()) fa->all_down(); down(); }
    } *const null = new P{0, 0, 0, 0, 0, 0}, pool[M], *pit = pool;

    void rot(P* o) {
        bool dd = o->d();
        P *f = o->fa, *t = o->c(!dd);
        if (f->has_fa()) f->fa->c(f->d()) = o; o->fa = f->fa;
        if (t != null) t->fa = f; f->c(dd) = t;
        o->c(!dd) = f->up(); f->fa = o;
    }
    void splay(P* o) {
        o->all_down();
        while (o->has_fa()) {
            if (o->fa->has_fa())
                rot(o->d() ^ o->fa->d() ? o : o->fa);
            rot(o);
        }
        o->up();
    }
    void access(P* u, P* v = null) {
        if (u == null) return;
        splay(u); u->rs = v;
        access(u->up()->fa, u);
    }
    void make_root(P* o) {
        access(o); splay(o); o->do_rev();
    }
    void split(P* o, P* u) {
        make_root(o); access(u); splay(u);
    }
    void link(P* u, P* v) {
        make_root(u); u->fa = v;
    }
    void cut(P* u, P* v) {
        split(u, v);
        u->fa = v->ls = null; v->up();
    }
```

```
    bool adj(P* u, P* v) {
        split(u, v);
        return v->ls == u && u->ls == null && u->rs == null;
    }
    bool linked(P* u, P* v) {
        split(u, v);
        return u == v || u->fa != null;
    }
    P* findrt(P* o) {
        access(o); splay(o);
        while (o->ls != null) o = o->ls;
        return o;
    }
    P* findfa(P* rt, P* u) {
        split(rt, u);
        u = u->ls;
        while (u->rs != null) {
            u = u->rs;
            u->down();
        }
        return u;
    }
}
```

- 维护子树大小

```
P* up() {
    sz = ls->sz + rs->sz + _sz + 1;
    return this;
}
void access(P* u, P* v = null) {
    if (u == null) return;
    splay(u);
    u->_sz += u->rs->sz - v->sz;
    u->rs = v;
    access(u->up()->fa, u);
}
void link(P* u, P* v) {
    split(u, v);
    u->fa = v; v->_sz += u->sz;
    v->up();
}
```

### 莫队

- [l, r)

```
while (l > q.l) mv(--l, 1);
while (r < q.r) mv(r++, 1);
while (l < q.l) mv(l++, -1);
while (r > q.r) mv(--r, -1);
```

- 树上莫队
- 注意初始状态 u = v = 1, flip(1)

```
struct Q {
    int u, v, idx;
    bool operator < (const Q& b) const {
        const Q& a = *this;
        return blk[a.u] < blk[b.u] || (blk[a.u] == blk[b.u] && in[a.v] < in[b.v]);
    }
};

void dfs(int u = 1, int d = 0) {
    static int S[maxn], sz = 0, blk_cnt = 0, clk = 0;
    in[u] = clk++;
    dep[u] = d;
    int btm = sz;
    for (int v: G[u]) {
        if (v == fa[u]) continue;
        fa[v] = u;
```

```
        dfs(v, d + 1);
        if (sz - btm >= B) {
            while (sz > btm) blk[S[--sz]] = blk_cnt;
            ++blk_cnt;
        }
    }
    S[sz++] = u;
    if (u == 1) while (sz) blk[S[--sz]] = blk_cnt - 1;
}

void flip(int k) {
    dbg(k);
    if (vis[k]) {
        // ...
    } else {
        // ...
    }
    vis[k] ^= 1;
}

void go(int& k) {
    if (bug == -1) {
        if (vis[k] && !vis[fa[k]]) bug = k;
        if (!vis[k] && vis[fa[k]]) bug = fa[k];
    }
    flip(k);
    k = fa[k];
}

void mv(int a, int b) {
    bug = -1;
    if (vis[b]) bug = b;
    if (dep[a] < dep[b]) swap(a, b);
    while (dep[a] > dep[b]) go(a);
    while (a != b) {
        go(a); go(b);
    }
    go(a); go(bug);
}

for (Q& q: query) {
    mv(u, q.u); u = q.u;
    mv(v, q.v); v = q.v;
    ans[q.idx] = Ans;
}
```

# 数学

## 矩阵运算

```
struct Mat {
    static const LL M = 2;
    LL v[M][M];
    Mat() { memset(v, 0, sizeof v); }
    void eye() { FOR (i, 0, M) v[i][i] = 1; }
    LL* operator [] (LL x) { return v[x]; }
    const LL* operator [] (LL x) const { return v[x]; }
    Mat operator * (const Mat& B) {
        const Mat& A = *this;
        Mat ret;
        FOR (k, 0, M)
            FOR (i, 0, M) if (A[i][k])
                FOR (j, 0, M)
                    ret[i][j] = (ret[i][j] + A[i][k] * B[k][j]) % MOD;
        return ret;
    }
    Mat pow(LL n) const {
        Mat A = *this, ret; ret.eye();
        for (; n; n >>= 1, A = A * A)
            if (n & 1) ret = ret * A;
```

```cpp
        return ret;
    }
    Mat operator + (const Mat& B) {
        const Mat& A = *this;
        Mat ret;
        FOR (i, 0, M)
            FOR (j, 0, M)
                 ret[i][j] = (A[i][j] + B[i][j]) % MOD;
        return ret;
    }
    void prt() const {
        FOR (i, 0, M)
            FOR (j, 0, M)
                 printf("%lld%c", (*this)[i][j], j == M - 1 ? '\n' : ' ');
    }
};
```

## 筛

- 线性筛

```cpp
const LL p_max = 1E6 + 100;
LL pr[p_max], p_sz;
void get_prime() {
    static bool vis[p_max];
    FOR (i, 2, p_max) {
        if (!vis[i]) pr[p_sz++] = i;
        FOR (j, 0, p_sz) {
            if (pr[j] * i >= p_max) break;
            vis[pr[j] * i] = 1;
            if (i % pr[j] == 0) break;
        }
    }
}
```

- 线性筛 + 欧拉函数

```cpp
const LL p_max = 1E5 + 100;
LL phi[p_max];
void get_phi() {
    phi[1] = 1;
    static bool vis[p_max];
    static LL prime[p_max], p_sz, d;
    FOR (i, 2, p_max) {
        if (!vis[i]) {
            prime[p_sz++] = i;
            phi[i] = i - 1;
        }
        for (LL j = 0; j < p_sz && (d = i * prime[j]) < p_max; ++j) {
            vis[d] = 1;
            if (i % prime[j] == 0) {
                phi[d] = phi[i] * prime[j];
                break;
            }
            else phi[d] = phi[i] * (prime[j] - 1);
        }
    }
}
```

- 线性筛 + 莫比乌斯函数

```cpp
const LL p_max = 1E5 + 100;
LL mu[p_max];
void get_mu() {
    mu[1] = 1;
    static bool vis[p_max];
    static LL prime[p_max], p_sz, d;
    FOR (i, 2, p_max) {
        if (!vis[i]) {
            prime[p_sz++] = i;
            mu[i] = -1;
        }
```

```cpp
        for (LL j = 0; j < p_sz && (d = i * prime[j]) < p_max; ++j) {
            vis[d] = 1;
            if (i % prime[j] == 0) {
                mu[d] = 0;
                break;
            }
            else mu[d] = -mu[i];
        }
    }
}
```

### 亚线性筛

#### min_25

```cpp
namespace min25 {
    const int M = 1E6 + 100;
    LL B, N;

    // g(x)
    inline LL pg(LL x) { return 1; }
    inline LL ph(LL x) { return x % MOD; }
    // Sum[g(i),{x,2,x}]
    inline LL psg(LL x) { return x % MOD - 1; }
    inline LL psh(LL x) {
        static LL inv2 = (MOD + 1) / 2;
        x %= MOD;
        return x * (x + 1) % MOD * inv2 % MOD - 1;
    }
    // f(pp=p^k)
    inline LL fpk(LL p, LL e, LL pp) { return (pp - pp / p) % MOD; }
    // f(p) = fgh(g(p), h(p))
    inline LL fgh(LL g, LL h) { return h - g; }

    LL pr[M], pc, sg[M], sh[M];
    void get_prime(LL n) {
        static bool vis[M]; pc = 0;
        FOR (i, 2, n + 1) {
            if (!vis[i]) {
                pr[pc++] = i;
                sg[pc] = (sg[pc - 1] + pg(i)) % MOD;
                sh[pc] = (sh[pc - 1] + ph(i)) % MOD;
            }
            FOR (j, 0, pc) {
                if (pr[j] * i > n) break;
                vis[pr[j] * i] = 1;
                if (i % pr[j] == 0) break;
            }
        }
    }

    LL w[M];
    LL id1[M], id2[M], h[M], g[M];
    inline LL id(LL x) { return x <= B ? id1[x] : id2[N / x]; }

    LL go(LL x, LL k) {
        if (x <= 1 || (k >= 0 && pr[k] > x)) return 0;
        LL t = id(x);
        LL ans = fgh((g[t] - sg[k + 1]), (h[t] - sh[k + 1]));
        FOR (i, k + 1, pc) {
            LL p = pr[i];
            if (p * p > x) break;
            ans -= fgh(pg(p), ph(p));
            for (LL pp = p, e = 1; pp <= x; ++e, pp = pp * p)
                ans += fpk(p, e, pp) * (1 + go(x / pp, i)) % MOD;
        }
        return ans % MOD;
    }

    LL solve(LL _N) {
        N = _N;
```

```
        B = sqrt(N + 0.5);
        get_prime(B);
        int sz = 0;
        for (LL l = 1, v, r; l <= N; l = r + 1) {
            v = N / l; r = N / v;
            w[sz] = v; g[sz] = psg(v); h[sz] = psh(v);
            if (v <= B) id1[v] = sz; else id2[r] = sz;
            sz++;
        }
        FOR (k, 0, pc) {
            LL p = pr[k];
            FOR (i, 0, sz) {
                LL v = w[i]; if (p * p > v) break;
                LL t = id(v / p);
                g[i] = (g[i] - (g[t] - sg[k]) * pg(p)) % MOD;
                h[i] = (h[i] - (h[t] - sh[k]) * ph(p)) % MOD;
            }
        }
        return (go(N, -1) % MOD + MOD + 1) % MOD;
    }
    pair<LL, LL> sump(LL l, LL r) {
        return {h[id(r)] - h[id(l - 1)],
                g[id(r)] - g[id(l - 1)]};
    }
}
```

**杜教筛**

求 $S(n) = \sum_{i=1}^{n} f(i)$，其中 $f$ 是一个积性函数。

构造一个积性函数 $g$，那么由 $(f * g)(n) = \sum_{d|n} f(d)g(\frac{n}{d})$，得到 $f(n) = (f * g)(n) - \sum_{d|n,d<n} f(d)g(\frac{n}{d})$。

$$
\begin{aligned}
g(1)S(n) &= \sum_{i=1}^{n}(f*g)(i) - \sum_{i=1}^{n}\sum_{d|i,d<i} f(d)g(\frac{n}{d}) && (1)\\
&\overset{t=\frac{i}{d}}{=} \sum_{i=1}^{n}(f*g)(i) - \sum_{t=2}^{n} g(t)S(\lfloor\frac{n}{t}\rfloor) && (2)
\end{aligned}
$$

当然，要能够由此计算 $S(n)$，会对 $f, g$ 提出一些要求：

- $f * g$ 要能够快速求前缀和。
- $g$ 要能够快速求分段和（前缀和）。
- 对于正常的积性函数 $g(1) = 1$，所以不会有什么问题。

在预处理 $S(n)$ 前 $n^{\frac{2}{3}}$ 项的情况下复杂度是 $O(n^{\frac{2}{3}})$。

```
namespace dujiao {
    const int M = 5E6;
    LL f[M] = {0, 1};
    void init() {
        static bool vis[M];
        static LL pr[M], p_sz, d;
        FOR (i, 2, M) {
            if (!vis[i]) { pr[p_sz++] = i; f[i] = -1; }
            FOR (j, 0, p_sz) {
                if ((d = pr[j] * i) >= M) break;
                vis[d] = 1;
                if (i % pr[j] == 0) {
                    f[d] = 0;
                    break;
                } else f[d] = -f[i];
            }
        }
        FOR (i, 2, M) f[i] += f[i - 1];
    }
    inline LL s_fg(LL n) { return 1; }
    inline LL s_g(LL n) { return n; }
```

```
    LL N, rd[M];
    bool vis[M];
    LL go(LL n) {
        if (n < M) return f[n];
        LL id = N / n;
        if (vis[id]) return rd[id];
        vis[id] = true;
        LL& ret = rd[id] = s_fg(n);
        for (LL l = 2, v, r; l <= n; l = r + 1) {
            v = n / l; r = n / v;
            ret -= (s_g(r) - s_g(l - 1)) * go(v);
        }
        return ret;
    }
    LL solve(LL n) {
        N = n;
        memset(vis, 0, sizeof vis);
        return go(n);
    }
}
```

### 素数测试

- 前置：快速乘、快速幂
- int 范围内只需检查 2, 7, 61
- long long 范围 2, 325, 9375, 28178, 450775, 9780504, 1795265022
- 3E15 内 2, 2570940, 880937, 610386380, 4130785767
- 4E13 内 2, 2570940, 211991001, 3749873356
- http://miller-rabin.appspot.com/

```
bool checkQ(LL a, LL n) {
    if (n == 2) return 1;
    if (n == 1 || !(n & 1)) return 0;
    LL d = n - 1;
    while (!(d & 1)) d >>= 1;
    LL t = bin(a, d, n);  // 不一定需要快速乘
    while (d != n - 1 && t != 1 && t != n - 1) {
        t = mul(t, t, n);
        d <<= 1;
    }
    return t == n - 1 || d & 1;
}

bool primeQ(LL n) {
    static vector<LL> t = {2, 325, 9375, 28178, 450775, 9780504, 1795265022};
    if (n <= 1) return false;
    for (LL k: t) if (!checkQ(k, n)) return false;
    return true;
}
```

## Pollard-Rho

```
mt19937 mt(time(0));
LL pollard_rho(LL n, LL c) {
    LL x = uniform_int_distribution<LL>(1, n - 1)(mt), y = x;
    auto f = [&](LL v) { LL t = mul(v, v, n) + c; return t < n ? t : t - n; };
    while (1) {
        x = f(x); y = f(f(y));
        if (x == y) return n;
        LL d = gcd(abs(x - y), n);
        if (d != 1) return d;
    }
}

LL fac[100], fcnt;
void get_fac(LL n, LL cc = 19260817) {
    if (n == 4) { fac[fcnt++] = 2; fac[fcnt++] = 2; return; }
    if (primeQ(n)) { fac[fcnt++] = n; return; }
```

```
    LL p = n;
    while (p == n) p = pollard_rho(n, --cc);
    get_fac(p); get_fac(n / p);
}

void go_fac(LL n) { fcnt = 0; if (n > 1) get_fac(n); }
```

## BM 线性递推

```
namespace BerlekampMassey {
    using V = vector<LL>;
    inline void up(LL& a, LL b) { (a += b) %= MOD; }
    V mul(const V&a, const V& b, const V& m, int k) {
        V r; r.resize(2 * k - 1);
        FOR (i, 0, k) FOR (j, 0, k) up(r[i + j], a[i] * b[j]);
        FORD (i, k - 2, -1) {
            FOR (j, 0, k) up(r[i + j], r[i + k] * m[j]);
            r.pop_back();
        }
        return r;
    }

    V pow(LL n, const V& m) {
        int k = (int) m.size() - 1; assert (m[k] == -1 || m[k] == MOD - 1);
        V r(k), x(k); r[0] = x[1] = 1;
        for (; n; n >>= 1, x = mul(x, x, m, k))
            if (n & 1) r = mul(x, r, m, k);
        return r;
    }

    LL go(const V& a, const V& x, LL n) {
        // a: (-1, a1, a2, ..., ak).reverse
        // x: x1, x2, ..., xk
        // x[n] = sum[a[i]*x[n-i],{i,1,k}]
        int k = (int) a.size() - 1;
        if (n <= k) return x[n - 1];
        if (a.size() == 2) return x[0] * bin(a[0], n - 1, MOD) % MOD;
        V r = pow(n - 1, a);
        LL ans = 0;
        FOR (i, 0, k) up(ans, r[i] * x[i]);
        return (ans + MOD) % MOD;
    }

    V BM(const V& x) {
        V C{-1}, B{-1};
        LL L = 0, m = 1, b = 1;
        FOR (n, 0, x.size()) {
            LL d = 0;
            FOR (i, 0, L + 1) up(d, C[i] * x[n - i]);
            if (d == 0) { ++m; continue; }
            V T = C;
            LL c = MOD - d * get_inv(b, MOD) % MOD;
            FOR (_, C.size(), B.size() + m) C.push_back(0);
            FOR (i, 0, B.size()) up(C[i + m], c * B[i]);
            if (2 * L > n) { ++m; continue; }
            L = n + 1 - L; B.swap(T); b = d; m = 1;
        }
        reverse(C.begin(), C.end());
        return C;
    }
}
```

## 扩展欧几里得

- 求 $ax + by = gcd(a, b)$ 的一组解
- 如果 $a$ 和 $b$ 互素，那么 $x$ 是 $a$ 在模 $b$ 下的逆元
- 注意 $x$ 和 $y$ 可能是负数

```
LL ex_gcd(LL a, LL b, LL &x, LL &y) {
    if (b == 0) { x = 1; y = 0; return a; }
```

```
    LL ret = ex_gcd(b, a % b, y, x);
    y -= a / b * x;
    return ret;
}
```

- 卡常欧几里得

```
inline int ctz(LL x) { return __builtin_ctzll(x); }
LL gcd(LL a, LL b) {
    if (!a) return b; if (!b) return a;
    int t = ctz(a | b);
    a >>= ctz(a);
    do {
        b >>= ctz(b);
        if (a > b) swap(a, b);
        b -= a;
    } while (b);
    return a << t;
}
```

### 类欧几里得

- $m = \lfloor \frac{an+b}{c} \rfloor$.
- $f(a,b,c,n) = \sum_{i=0}^{n} \lfloor \frac{ai+b}{c} \rfloor$: 当 $a \geq c$ or $b \geq c$ 时，$f(a,b,c,n) = (\frac{a}{c})n(n+1)/2 + (\frac{b}{c})(n+1) + f(a \bmod c, b \bmod c, c, n)$；否则 $f(a,b,c,n) = nm - f(c, c-b-1, a, m-1)$。
- $g(a,b,c,n) = \sum_{i=0}^{n} i \lfloor \frac{ai+b}{c} \rfloor$: 当 $a \geq c$ or $b \geq c$ 时，$g(a,b,c,n) = (\frac{a}{c})n(n+1)(2n+1)/6 + (\frac{b}{c})n(n+1)/2 + g(a \bmod c, b \bmod c, c, n)$；否则 $g(a,b,c,n) = \frac{1}{2}(n(n+1)m - f(c, c-b-1, a, m-1) - h(c, c-b-1, a, m-1))$。
- $h(a,b,c,n) = \sum_{i=0}^{n} \lfloor \frac{ai+b}{c} \rfloor^2$: 当 $a \geq c$ or $b \geq c$ 时，$h(a,b,c,n) = (\frac{a}{c})^2 n(n+1)(2n+1)/6 + (\frac{b}{c})^2 (n+1) + (\frac{a}{c})(\frac{b}{c})n(n+1) + h(a \bmod c, b \bmod c, c, n) + 2(\frac{a}{c})g(a \bmod c, b \bmod c, c, n) + 2(\frac{b}{c})f(a \bmod c, b \bmod c, c, n)$；否则 $h(a,b,c,n) = nm(m+1) - 2g(c, c-b-1, a, m-1) - 2f(c, c-b-1, a, m-1) - f(a,b,c,n)$。

### 逆元

- 如果 $p$ 不是素数，使用拓展欧几里得

- 前置模板：快速幂 / 扩展欧几里得

```
inline LL get_inv(LL x, LL p) { return bin(x, p - 2, p); }
LL get_inv(LL a, LL M) {
    static LL x, y;
    assert(exgcd(a, M, x, y) == 1);
    return (x % M + M) % M;
}
```

- 预处理 1~n 的逆元

```
LL inv[N];
void inv_init(LL n, LL p) {
    inv[1] = 1;
    FOR (i, 2, n)
        inv[i] = (p - p / i) * inv[p % i] % p;
}
```

- 预处理阶乘及其逆元

```
LL invf[M], fac[M] = {1};
void fac_inv_init(LL n, LL p) {
    FOR (i, 1, n)
        fac[i] = i * fac[i - 1] % p;
    invf[n - 1] = bin(fac[n - 1], p - 2, p);
    FORD (i, n - 2, -1)
        invf[i] = invf[i + 1] * (i + 1) % p;
}
```

### 组合数

- 如果数较小，模较大时使用逆元
- 前置模板：逆元-预处理阶乘及其逆元

```
inline LL C(LL n, LL m) { // n >= m >= 0
    return n < m || m < 0 ? 0 : fac[n] * invf[m] % MOD * invf[n - m] % MOD;
}
```

- 如果模数较小，数字较大，使用 Lucas 定理
- 前置模板可选 1：求组合数（如果使用阶乘逆元，需 `fac_inv_init(MOD, MOD);`）
- 前置模板可选 2：模数不固定下使用，无法单独使用。

```
LL C(LL n, LL m) { // m >= n >= 0
    if (m - n < n) n = m - n;
    if (n < 0) return 0;
    LL ret = 1;
    FOR (i, 1, n + 1)
        ret = ret * (m - n + i) % MOD * bin(i, MOD - 2, MOD) % MOD;
    return ret;
}
```

```
LL Lucas(LL n, LL m) { // m >= n >= 0
    return m ? C(n % MOD, m % MOD) * Lucas(n / MOD, m / MOD) % MOD : 1;
}
```

- 组合数预处理

```
LL C[M][M];
void init_C(int n) {
    FOR (i, 0, n) {
        C[i][0] = C[i][i] = 1;
        FOR (j, 1, i)
            C[i][j] = (C[i - 1][j] + C[i - 1][j - 1]) % MOD;
    }
}
```

## 斯特灵数

### 第一类斯特灵数

- 绝对值是 $n$ 个元素划分为 $k$ 个环排列的方案数。
- $s(n,k)=s(n-1,k-1)+(n-1)s(n-1,k)$

### 第二类斯特灵数

- $n$ 个元素划分为 $k$ 个等价类的方案数
- $S(n,k)=S(n-1,k-1)+kS(n-1,k)$

```
S[0][0] = 1;
FOR (i, 1, N)
    FOR (j, 1, i + 1) S[i][j] = (S[i - 1][j - 1] + j * S[i - 1][j]) % MOD;
```

## simpson 自适应积分

```
LD simpson(LD l, LD r) {
    LD c = (l + r) / 2;
    return (f(l) + 4 * f(c) + f(r)) * (r - l) / 6;
}

LD asr(LD l, LD r, LD eps, LD S) {
    LD m = (l + r) / 2;
    LD L = simpson(l, m), R = simpson(m, r);
    if (fabs(L + R - S) < 15 * eps) return L + R + (L + R - S) / 15;
    return asr(l, m, eps / 2, L) + asr(m, r, eps / 2, R);
}

LD asr(LD l, LD r, LD eps) { return asr(l, r, eps, simpson(l, r)); }
```

- FWT

```
template<typename T>
void fwt(LL a[], int n, T f) {
    for (int d = 1; d < n; d *= 2)
        for (int i = 0, t = d * 2; i < n; i += t)
```

```
            FOR (j, 0, d)
                f(a[i + j], a[i + j + d]);
}

auto f = [](LL& a, LL& b) { // xor
        LL x = a, y = b;
        a = (x + y) % MOD;
        b = (x - y + MOD) % MOD;
};
```

### 快速乘

```
LL mul(LL a, LL b, LL m) {
    LL ret = 0;
    while (b) {
        if (b & 1) {
            ret += a;
            if (ret >= m) ret -= m;
        }
        a += a;
        if (a >= m) a -= m;
        b >>= 1;
    }
    return ret;
}
```

- O(1)

```
LL mul(LL u, LL v, LL p) {
    return (u * v - LL((long double) u * v / p) * p + p) % p;
}
LL mul(LL u, LL v, LL p) { // 卡常
    LL t = u * v - LL((long double) u * v / p) * p;
    return t < 0 ? t + p : t;
}
```

### 快速幂

- 如果模数是素数，则可在函数体内加上 `n %= MOD - 1;`（费马小定理）。

```
LL bin(LL x, LL n, LL MOD) {
    LL ret = MOD != 1;
    for (x %= MOD; n; n >>= 1, x = x * x % MOD)
        if (n & 1) ret = ret * x % MOD;
    return ret;
}
```

- 防爆 LL
- 前置模板：快速乘

```
LL bin(LL x, LL n, LL MOD) {
    LL ret = MOD != 1;
    for (x %= MOD; n; n >>= 1, x = mul(x, x, MOD))
        if (n & 1) ret = mul(ret, x, MOD);
    return ret;
}
```

### 高斯消元

#### 简略版（只要唯一解，不需要求解自由变量的个数）

```
const double EPS = 1E-9;
int n;
vector<vector<double> > a(n, vector<double>(n));

double det = 1;
for (int i = 0; i < n; ++i) {
  int k = i;
  for (int j = i + 1; j < n; ++j)
    if (abs(a[j][i]) > abs(a[k][i])) k = j;
```

```
  if (abs(a[k][i]) < EPS) {
    det = 0;
    break;
  }
  swap(a[i], a[k]);
  if (i != k) det = -det;
  det *= a[i][i];
  for (int j = i + 1; j < n; ++j) a[i][j] /= a[i][i];
  for (int j = 0; j < n; ++j)
    if (j != i && abs(a[j][i]) > EPS)
      for (int k = i + 1; k < n; ++k) a[j][k] -= a[i][k] * a[j][i];
}

cout << det;
```

### 完整版

- n - 方程个数，m - 变量个数，a 是 n * (m + 1) 的增广矩阵，free 是否为自由变量
- 返回自由变量个数，-1 无解
- 浮点数版本

```
typedef double LD;
const LD eps = 1E-10;
const int maxn = 2000 + 10;

int n, m;
LD a[maxn][maxn], x[maxn];
bool free_x[maxn];

inline int sgn(LD x) { return (x > eps) - (x < -eps); }

int gauss(LD a[maxn][maxn], int n, int m) {
    memset(free_x, 1, sizeof free_x); memset(x, 0, sizeof x);
    int r = 0, c = 0;
    while (r < n && c < m) {
        int m_r = r;
        FOR (i, r + 1, n)
            if (fabs(a[i][c]) > fabs(a[m_r][c])) m_r = i;
        if (m_r != r)
            FOR (j, c, m + 1)
                 swap(a[r][j], a[m_r][j]);
        if (!sgn(a[r][c])) {
            a[r][c] = 0;
            ++c;
            continue;
        }
        FOR (i, r + 1, n)
            if (a[i][c]) {
                LD t = a[i][c] / a[r][c];
                FOR (j, c, m + 1) a[i][j] -= a[r][j] * t;
            }
        ++r; ++c;
    }
    FOR (i, r, n)
        if (sgn(a[i][m])) return -1;
    if (r < m) {
        FORD (i, r - 1, -1) {
            int f_cnt = 0, k = -1;
            FOR (j, 0, m)
                if (sgn(a[i][j]) && free_x[j]) {
                    ++f_cnt;
                    k = j;
                }
            if(f_cnt > 0) continue;
            LD s = a[i][m];
            FOR (j, 0, m)
                if (j != k) s -= a[i][j] * x[j];
            x[k] = s / a[i][k];
            free_x[k] = 0;
```

```
        }
        return m - r;
    }
    FORD (i, m - 1, -1) {
        LD s = a[i][m];
        FOR (j, i + 1, m)
            s -= a[i][j] * x[j];
        x[i] = s / a[i][i];
    }
    return 0;
}
```

- 数据

```
3 4
1 1 -2 2
2 -3 5 1
4 -1 1 5
5 0 -1 7
// many

3 4
1 1 -2 2
2 -3 5 1
4 -1 -1 5
5 0 -1 0 2
// no

3 4
1 1 -2 2
2 -3 5 1
4 -1 1 5
5 0 1 0 7
// one
```

### 质因数分解

hkk 注：感觉像这样枚举 sqrt 以内的所有质数很慢……hkk 一般是在筛素数的时候记录下每个数的最小质因子，然后不断地除下去，复杂度 $O(\log)$。

- 前置模板：素数筛
- 带指数

```
LL factor[30], f_sz, factor_exp[30];
void get_factor(LL x) {
    f_sz = 0;
    LL t = sqrt(x + 0.5);
    for (LL i = 0; pr[i] <= t; ++i)
        if (x % pr[i] == 0) {
            factor_exp[f_sz] = 0;
            while (x % pr[i] == 0) {
                x /= pr[i];
                ++factor_exp[f_sz];
            }
            factor[f_sz++] = pr[i];
        }
    if (x > 1) {
        factor_exp[f_sz] = 1;
        factor[f_sz++] = x;
    }
}
```

- 不带指数

```
LL factor[30], f_sz;
```

```
void get_factor(LL x) {
    f_sz = 0;
    LL t = sqrt(x + 0.5);
    for (LL i = 0; pr[i] <= t; ++i)
        if (x % pr[i] == 0) {
            factor[f_sz++] = pr[i];
            while (x % pr[i] == 0) x /= pr[i];
        }
    if (x > 1) factor[f_sz++] = x;
}
```

## 原根

- 定义：设 m 是正整数，a 是整数，若 a 模 m 的阶等于 φ(m)，则称 a 为模 m 的一个原根。（其中 φ(m) 表示 m 的欧拉函数）
- 性质：假设一个数 $g$ 是 $P$ 的原根，那么 $g^i \bmod P$ 的结果两两不同，且有 $1 < g < P$，$0 < i < P$，归根到底就是 $g^{P-1} = 1 \pmod P$ 当且仅当指数为 $P-1$ 的时候成立 (这里 $P$ 是素数)。简单来说，$g^i \bmod p \neq g^j \bmod p$（$p$ 为素数），其中 $i \neq j$ 且 $i, j$ 介于 1 至 $p-1$ 之间，则 $g$ 为 $p$ 的原根。
- 前置模板：素数筛，快速幂，分解质因数
- 要求 p 为质数

```
LL find_smallest_primitive_root(LL p) {
    get_factor(p - 1);
    FOR (i, 2, p) {
        bool flag = true;
        FOR (j, 0, f_sz)
            if (bin(i, (p - 1) / factor[j], p) == 1) {
                flag = false;
                break;
            }
        if (flag) return i;
    }
    assert(0); return -1;
}
```

## 公式

### 一些数论公式

- 当 $x \geq \phi(p)$ 时有 $a^x \equiv a^{x \ mod \ \phi(p) + \phi(p)} \pmod p$
- $\mu^2(n) = \sum_{d^2|n} \mu(d)$
- $\sum_{d|n} \varphi(d) = n$
- $\sum_{d|n} 2^{\omega(d)} = \sigma_0(n^2)$，其中 $\omega$ 是不同素因子个数
- $\sum_{d|n} \mu^2(d) = 2^{\omega(d)}$

### 一些数论函数求和的例子

- $\sum_{i=1}^n i[gcd(i,n)=1] = \frac{n\varphi(n)+[n=1]}{2}$
- $\sum_{i=1}^n \sum_{j=1}^m [gcd(i,j)=x] = \sum_d \mu(d) \lfloor \frac{n}{dx} \rfloor \lfloor \frac{m}{dx} \rfloor$
- $\sum_{i=1}^n \sum_{j=1}^m gcd(i,j) = \sum_{i=1}^n \sum_{j=1}^m \sum_{d|gcd(i,j)} \varphi(d) = \sum_d \varphi(d) \lfloor \frac{n}{d} \rfloor \lfloor \frac{m}{d} \rfloor$
- $S(n) = \sum_{i=1}^n \mu(i) = 1 - \sum_{i=1}^n \sum_{d|i,d<i} \mu(d) \overset{t=\frac{i}{d}}{=} 1 - \sum_{t=2}^n S(\lfloor \frac{n}{t} \rfloor)$
  - 利用 $[n=1] = \sum_{d|n} \mu(d)$
- $S(n) = \sum_{i=1}^n \varphi(i) = \sum_{i=1}^n i - \sum_{i=1}^n \sum_{d|i,d<i} \varphi(i) \overset{t=\frac{i}{d}}{=} \frac{i(i+1)}{2} - \sum_{t=2}^n S(\frac{n}{t})$
  - 利用 $n = \sum_{d|n} \varphi(d)$
- $\sum_{i=1}^n \mu^2(i) = \sum_{i=1}^n \sum_{d^2|n} \mu(d) = \sum_{d=1}^{\lfloor \sqrt{n} \rfloor} \mu(d) \lfloor \frac{n}{d^2} \rfloor$
- $\sum_{i=1}^n \sum_{j=1}^n gcd^2(i,j) = \sum_d d^2 \sum_t \mu(t) \lfloor \frac{n}{dt} \rfloor^2$
  $\overset{x=dt}{=} \sum_x \lfloor \frac{n}{x} \rfloor^2 \sum_{d|x} d^2 \mu(\frac{x}{d})$
- $\sum_{i=1}^n \varphi(i) = \frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n [i \perp j] - 1 = \frac{1}{2} \sum_{i=1}^n \mu(i) \cdot \lfloor \frac{n}{i} \rfloor^2 - 1$

**斐波那契数列性质**

- $F_{a+b} = F_{a-1} \cdot F_b + F_a \cdot F_{b+1}$
- $F_1 + F_3 + \cdots + F_{2n-1} = F_{2n}, F_2 + F_4 + \cdots + F_{2n} = F_{2n+1} - 1$
- $\sum_{i=1}^{n} F_i = F_{n+2} - 1$
- $\sum_{i=1}^{n} F_i^2 = F_n \cdot F_{n+1}$
- $F_n^2 = (-1)^{n-1} + F_{n-1} \cdot F_{n+1}$
- $gcd(F_a, F_b) = F_{gcd(a,b)}$
- 模 $n$ 周期（皮萨诺周期）
  - $\pi(p^k) = p^{k-1}\pi(p)$
  - $\pi(nm) = lcm(\pi(n), \pi(m)), \forall n \perp m$
  - $\pi(2) = 3, \pi(5) = 20$
  - $\forall p \equiv \pm 1 \pmod{10}, \pi(p) | p - 1$
  - $\forall p \equiv \pm 2 \pmod{5}, \pi(p) | 2p + 2$

**常见生成函数**

- $(1 + ax)^n = \sum_{k=0}^{n} \binom{n}{k} a^k x^k$
- $\dfrac{1 - x^{r+1}}{1 - x} = \sum_{k=0}^{n} x^k$
- $\dfrac{1}{1 - ax} = \sum_{k=0}^{\infty} a^k x^k$
- $\dfrac{1}{(1 - x)^2} = \sum_{k=0}^{\infty} (k + 1) x^k$
- $\dfrac{1}{(1 - x)^n} = \sum_{k=0}^{\infty} \binom{n+k-1}{k} x^k$
- $e^x = \sum_{k=0}^{\infty} \dfrac{x^k}{k!}$
- $\ln(1 + x) = \sum_{k=0}^{\infty} \dfrac{(-1)^{k+1}}{k} x^k$

**佩尔方程**

若一个丢番图方程具有以下的形式：$x^2 - ny^2 = 1$。且 $n$ 为正整数，则称此二元二次不定方程为**佩尔方程**。

若 $n$ 是完全平方数，则这个方程式只有平凡解 $(\pm 1, 0)$（实际上对任意的 $n$，$(\pm 1, 0)$ 都是解）。对于其余情况，拉格朗日证明了佩尔方程总有非平凡解。而这些解可由 $\sqrt{n}$ 的连分数求出。

$$x = [a_0; a_1, a_2, a_3] = x = a_0 + \cfrac{1}{a_1 + \cfrac{1}{a_2 + \cfrac{1}{a_3 + \cfrac{1}{\ddots}}}}$$

设 $\frac{p_i}{q_i}$ 是 $\sqrt{n}$ 的连分数表示: $[a_0; a_1, a_2, a_3, \ldots]$ 的渐近分数列, 由连分数理论知存在 $i$ 使得 $(p_i, q_i)$ 为佩尔方程的解。取其中最小的 $i$, 将对应的 $(p_i, q_i)$ 称为佩尔方程的基本解, 或最小解, 记作 $(x_1, y_1)$, 则所有的解 $(x_i, y_i)$ 可表示成如下形式: $x_i + y_i\sqrt{n} = (x_1 + y_1\sqrt{n})^i$。或者由以下的递回关系式得到:

$x_{i+1} = x_1 x_i + n y_1 y_i, y_{i+1} = x_1 y_i + y_1 x_i$。

**但是：** 佩尔方程千万不要去推（虽然推起来很有趣，但结果不一定好看，会是两个式子）。记住佩尔方程结果的形式通常是 $a_n = ka_{n-1} - a_{n-2}$（$a_{n-2}$ 前的系数通常是 $-1$）。暴力 / 凑出两个基础解之后加上一个 $0$，容易解出 $k$ 并验证。

**Burnside & Polya**

- $|X/G| = \frac{1}{|G|} \sum_{g \in G} |X^g|$

注：$X^g$ 是 $g$ 下的不动点数量，也就是说有多少种东西用 $g$ 作用之后可以保持不变。

- $|Y^X/G| = \frac{1}{|G|} \sum_{g \in G} m^{c(g)}$

注：用 $m$ 种颜色染色，然后对于某一种置换 $g$，有 $c(g)$ 个置换环，为了保证置换后颜色仍然相同，每个置换环必须染成同色。

**皮克定理**

$2S = 2a + b - 2$

- $S$ 多边形面积
- $a$ 多边形内部点数
- $b$ 多边形边上点数

**莫比乌斯反演**

- $g(n) = \sum_{d|n} f(d) \Leftrightarrow f(n) = \sum_{d|n} \mu(d) g(\frac{n}{d})$
- $f(n) = \sum_{n|d} g(d) \Leftrightarrow g(n) = \sum_{n|d} \mu(\frac{d}{n}) f(d)$

**低阶等幂求和**

- $\sum_{i=1}^{n} i^{1} = \frac{n(n+1)}{2} = \frac{1}{2}n^2 + \frac{1}{2}n$
- $\sum_{i=1}^{n} i^{2} = \frac{n(n+1)(2n+1)}{6} = \frac{1}{3}n^3 + \frac{1}{2}n^2 + \frac{1}{6}n$
- $\sum_{i=1}^{n} i^{3} = \left[\frac{n(n+1)}{2}\right]^2 = \frac{1}{4}n^4 + \frac{1}{2}n^3 + \frac{1}{4}n^2$
- $\sum_{i=1}^{n} i^{4} = \frac{n(n+1)(2n+1)(3n^2+3n-1)}{30} = \frac{1}{5}n^5 + \frac{1}{2}n^4 + \frac{1}{3}n^3 - \frac{1}{30}n$
- $\sum_{i=1}^{n} i^{5} = \frac{n^2(n+1)^2(2n^2+2n-1)}{12} = \frac{1}{6}n^6 + \frac{1}{2}n^5 + \frac{5}{12}n^4 - \frac{1}{12}n^2$

**一些组合公式**

- 错排公式：$D_1 = 0, D_2 = 1, D_n = (n-1)(D_{n-1} + D_{n-2}) = n!(\frac{1}{2!} - \frac{1}{3!} + \cdots + (-1)^n \frac{1}{n!}) = \lfloor \frac{n!}{e} + 0.5 \rfloor$
- 卡塔兰数（$n$ 对括号合法方案数，$n$ 个结点二叉树个数，$n \times n$ 方格中对角线下方的单调路径数，凸 $n+2$ 边形的三角形划分数，$n$ 个元素的合法出栈序列数）：$C_n = \frac{1}{n+1}\binom{2n}{n} = \frac{(2n)!}{(n+1)!n!}$

## 二次剩余

- 定义：$X^2$ 在数论中，特别在同余理论里，一个整数 $X$ 对另一个整数 $p$ 的**二次剩余**（英语：Quadratic residue）指 $X$ 的平方除以 $p$ 得到的余数。当存在某个 $X$，式子 $x^2 \equiv d \pmod p$ 成立时，称 $d$ 是模 $p$ 的二次剩余。

URAL 1132

```cpp
LL q1, q2, w;
struct P { // x + y * sqrt(w)
    LL x, y;
};

P pmul(const P& a, const P& b, LL p) {
    P res;
    res.x = (a.x * b.x + a.y * b.y % p * w) % p;
    res.y = (a.x * b.y + a.y * b.x) % p;
    return res;
}

P bin(P x, LL n, LL MOD) {
    P ret = {1, 0};
    for (; n; n >>= 1, x = pmul(x, x, MOD))
        if (n & 1) ret = pmul(ret, x, MOD);
    return ret;
}
LL Legendre(LL a, LL p) { return bin(a, (p - 1) >> 1, p); }

LL equation_solve(LL b, LL p) {
    if (p == 2) return 1;
    if ((Legendre(b, p) + 1) % p == 0)
        return -1;
    LL a;
    while (true) {
        a = rand() % p;
```

```
        w = ((a * a - b) % p + p) % p;
        if ((Legendre(w, p) + 1) % p == 0)
            break;
    }
    return bin({a, 1}, (p + 1) >> 1, p).x;
}

int main() {
    int T; cin >> T;
    while (T--) {
        LL a, p; cin >> a >> p;
        a = a % p;
        LL x = equation_solve(a, p);
        if (x == -1) {
            puts("No root");
        } else {
            LL y = p - x;
            if (x == y) cout << x << endl;
            else cout << min(x, y) << " " << max(x, y) << endl;
        }
    }
}
```

### （扩展）中国剩余定理

- hkk 版

```
ll smod(ll x, ll P) { return x < 0 ? x + P : (x >= P ? x - P : x); }
ll fmul(ll x, ll y, ll P) {
    ll ans = 0;
    x = smod(x, P);
    for (; y; y >>= 1, x = smod(x + x, P))
        if (y & 1) ans = smod(ans + x, P);
    return ans;
}
ll exgcd(ll a, ll b, ll &x, ll &y) {
    if (!b) return x = 1, y = 0, a;
    ll t = exgcd(b, a % b, y, x);
    y -= a / b * x;
    return t;
}
ll exCRT() {
    ll P = p[1], ans = a[1];
    for (int i = 2; i <= n; ++i) {
        ll a1 = P, a2 = p[i], c = smod((a[i] - ans) % a2 + a2, a2);
        ll x, y, d = exgcd(a1, a2, x, y);
        if (a1 % d) return -1;
        x = fmul(x, c / d, a2 / d);
        ans += x * P;
        P *= a2 / d;
        ans = smod(ans % P + P, P);
    }
    return ans;
}
```

- 模板原版
- 无解返回 -1
- 前置模板：扩展欧几里得

```
LL CRT(LL *m, LL *r, LL n) {
    if (!n) return 0;
    LL M = m[0], R = r[0], x, y, d;
    FOR (i, 1, n) {
        d = ex_gcd(M, m[i], x, y);
        if ((r[i] - R) % d) return -1;
        x = (r[i] - R) / d * x % (m[i] / d);
        // 防爆 LL
        // x = mul((r[i] - R) / d, x, m[i] / d);
        R += x * M;
```

```
        M = M / d * m[i];
        R %= M;
    }
    return R >= 0 ? R : R + M;
}
```

- 这儿还有一份 py 版，可以用高精度

```python
def exgcd(a, b):
    if not b:
        return a, 1, 0
    d, x, y = exgcd(b, a % b)
    return d, y, x - a // b * y

n = int(input())
m, a = map(int, input().split())
for i in range(1, n):
    m2, a2 = map(int, input().split())
    d, x, y = exgcd(m, m2)
    x *= (a2 - a) // d
    a += x * m
    m = m // d * m2
    a = ((a % m) + m) % m

print(a)
```

### 伯努利数和等幂求和

- 预处理逆元
- 预处理组合数
- $\sum_{i=0}^{n} i^k = \frac{1}{k+1} \sum_{i=0}^{k} \binom{k+1}{i} B_{k+1-i} (n+1)^i$.
- 也可以 $\sum_{i=0}^{n} i^k = \frac{1}{k+1} \sum_{i=0}^{k} \binom{k+1}{i} B^+_{k+1-i} n^i$。区别在于 $B^+_1 = 1/2$。(心态崩了)

```cpp
namespace Bernoulli {
    const int M = 100;
    LL inv[M] = {-1, 1};
    void inv_init(LL n, LL p) {
        FOR (i, 2, n)
            inv[i] = (p - p / i) * inv[p % i] % p;
    }

    LL C[M][M];
    void init_C(int n) {
        FOR (i, 0, n) {
            C[i][0] = C[i][i] = 1;
            FOR (j, 1, i)
                C[i][j] = (C[i - 1][j] + C[i - 1][j - 1]) % MOD;
        }
    }

    LL B[M] = {1};
    void init() {
        inv_init(M, MOD);
        init_C(M);
        FOR (i, 1, M - 1) {
            LL& s = B[i] = 0;
            FOR (j, 0, i)
                s += C[i + 1][j] * B[j] % MOD;
            s = (s % MOD * -inv[i + 1] % MOD + MOD) % MOD;
        }
    }

    LL p[M] = {1};
    LL go(LL n, LL k) {
        n %= MOD;
        if (k == 0) return n;
        FOR (i, 1, k + 2)
            p[i] = p[i - 1] * (n + 1) % MOD;
        LL ret = 0;
        FOR (i, 1, k + 2)
```

```
            ret += C[k + 1][i] * B[k + 1 - i] % MOD * p[i] % MOD;
        ret = ret % MOD * inv[k + 1] % MOD;
        return ret;
    }
}
```

### 单纯形

- 要求有基本解，也就是 x 为零向量可行
- v 要初始化为 0，n 表示向量长度，m 表示约束个数

```
// min{ b x } / max { c x }
// A x >= c   / A x <= b
// x >= 0
namespace lp {
    int n, m;
    double a[M][N], b[M], c[N], v;

    void pivot(int l, int e) {
        b[l] /= a[l][e];
        FOR (j, 0, n) if (j != e) a[l][j] /= a[l][e];
        a[l][e] = 1 / a[l][e];

        FOR (i, 0, m)
            if (i != l && fabs(a[i][e]) > 0) {
                b[i] -= a[i][e] * b[l];
                FOR (j, 0, n)
                    if (j != e) a[i][j] -= a[i][e] * a[l][j];
                a[i][e] = -a[i][e] * a[l][e];
            }
        v += c[e] * b[l];
        FOR (j, 0, n) if (j != e) c[j] -= c[e] * a[l][j];
        c[e] = -c[e] * a[l][e];
    }
    double simplex() {
        while (1) {
            v = 0;
            int e = -1, l = -1;
            FOR (i, 0, n) if (c[i] > eps) { e = i; break; }
            if (e == -1) return v;
            double t = INF;
            FOR (i, 0, m)
                if (a[i][e] > eps && t > b[i] / a[i][e]) {
                    t = b[i] / a[i][e];
                    l = i;
                }
            if (l == -1) return INF;
            pivot(l, e);
        }
    }
}
```

### 离散对数

#### BSGS

- 模数为素数

```
LL BSGS(LL a, LL b, LL p) { // a^x = b (mod p)
    a %= p;
    if (!a && !b) return 1;
    if (!a) return -1;
    static map<LL, LL> mp; mp.clear();
    LL m = sqrt(p + 1.5);
    LL v = 1;
    FOR (i, 1, m + 1) {
        v = v * a % p;
        mp[v * b % p] = i;
    }
    LL vv = v;
```

```
    FOR (i, 1, m + 1) {
        auto it = mp.find(vv);
        if (it != mp.end()) return i * m - it->second;
        vv = vv * v % p;
    }
    return -1;
}
```

### exBSGS

- 模数可以非素数

```
LL exBSGS(LL a, LL b, LL p) { // a^x = b (mod p)
    a %= p; b %= p;
    if (a == 0) return b > 1 ? -1 : b == 0 && p != 1;
    LL c = 0, q = 1;
    while (1) {
        LL g = __gcd(a, p);
        if (g == 1) break;
        if (b == 1) return c;
        if (b % g) return -1;
        ++c; b /= g; p /= g; q = a / g * q % p;
    }
    static map<LL, LL> mp; mp.clear();
    LL m = sqrt(p + 1.5);
    LL v = 1;
    FOR (i, 1, m + 1) {
        v = v * a % p;
        mp[v * b % p] = i;
    }
    FOR (i, 1, m + 1) {
        q = q * v % p;
        auto it = mp.find(q);
        if (it != mp.end()) return i * m - it->second + c;
    }
    return -1;
}
```

## 数论分块

$f(i) = \lfloor \frac{n}{i} \rfloor = v$ 时 $i$ 的取值范围是 $[l, r]$。

```
for (LL l = 1, v, r; l <= N; l = r + 1) {
    v = N / l; r = N / v;
}
```

## 博弈

- Nim 游戏：每轮从若干堆石子中的一堆取走若干颗。先手必胜条件为石子数量异或和非零。
- 阶梯 Nim 游戏：可以选择阶梯上某一堆中的若干颗向下推动一级，直到全部推下去。先手必胜条件是奇数阶梯的异或和非零（对于偶数阶梯的操作可以模仿）。
- Anti-SG：无法操作者胜。先手必胜的条件是：
  - SG 不为 0 且某个单一游戏的 SG 大于 1 。
  - SG 为 0 且没有单一游戏的 SG 大于 1。
- Every-SG：对所有单一游戏都要操作。先手必胜的条件是单一游戏中的最大 step 为奇数。
  - 对于终止状态 step 为 0
  - 对于 SG 为 0 的状态，step 是最大后继 step +1
  - 对于 SG 非 0 的状态，step 是最小后继 step +1
- 树上删边：叶子 SG 为 0，非叶子结点为所有子结点的 SG 值加 1 后的异或和。

尝试：

- 打表找规律
- 寻找一类必胜态（如对称局面）
- 直接博弈 dp

### 线性基

```cpp
constexpr int M = 64;
ull b[M];
bool ins(ull x) {
    for (int i = M - 1; ~i; --i) if ((x >> i) & 1) {
        if (b[i]) x ^= b[i];
        else { b[i] = x; return true; }
    }
    return false;
}
ull getmax() {
    ull ans = 0;
    for (int i = M - 1; ~i; --i) smax(ans, ans ^ b[i]);
    return ans;
}
```

### 多项式相关

#### NTT

- hkk 版

```cpp
void ntt(int *a, int n, int f = 1) { // a 是要 ntt 的序列, n 是长度（2^l), f 是正向运算 (1) 还是逆向运算（-1)
    for (int i = 0, j = 0; i < n; ++i) {
        if (i < j) std::swap(a[i], a[j]);
        for (int l = n >> 1; (j ^= l) < l; l >>= 1) ;
    }
    for (int i = 1; i < n; i <<= 1) {
        int w = fpow(f > 0 ? G : Gi, (P - 1) / (i << 1));
        for (int j = 0; j < n; j += (i << 1))
            for (int k = 0, e = 1; k < i; ++k, e = (ll)e * w % P) {
                int x = a[j + k], y = (ll)e * a[i + j + k] % P;
                a[j + k] = smod(x + y), a[i + j + k] = smod(x - y + P);
            }
    }
    if (f < 0) for (int i = 0, p = fpow(n, P - 2); i < n; ++i) a[i] = (ll)a[i] * p % P;
}
```

- 模板原版

```cpp
LL wn[N << 2], rev[N << 2];
int NTT_init(int n_) {
    int step = 0; int n = 1;
    for ( ; n < n_; n <<= 1) ++step;
    FOR (i, 1, n)
        rev[i] = (rev[i >> 1] >> 1) | ((i & 1) << (step - 1));
    int g = bin(G, (MOD - 1) / n, MOD);
    wn[0] = 1;
    for (int i = 1; i <= n; ++i)
        wn[i] = wn[i - 1] * g % MOD;
    return n;
}

void NTT(LL a[], int n, int f) {
    FOR (i, 0, n) if (i < rev[i])
        std::swap(a[i], a[rev[i]]);
    for (int k = 1; k < n; k <<= 1) {
        for (int i = 0; i < n; i += (k << 1)) {
            int t = n / (k << 1);
            FOR (j, 0, k) {
                LL w = f == 1 ? wn[t * j] : wn[n - t * j];
                LL x = a[i + j];
                LL y = a[i + j + k] * w % MOD;
                a[i + j] = (x + y) % MOD;
                a[i + j + k] = (x - y + MOD) % MOD;
            }
        }
    }
    if (f == -1) {
        LL ninv = get_inv(n, MOD);
        FOR (i, 0, n)
```

```
            a[i] = a[i] * ninv % MOD;
    }
}
```

### FFT

- n 需补成 2 的幂（n 必须超过 a 和 b 的最高指数之和）

```
struct Complex {
    double x, y;
    Complex(double x = 0, double y = 0) : x(x), y(y) { }
    Complex operator + (const Complex &a) const { return Complex(x + a.x, y + a.y); }
    Complex operator - (const Complex &a) const { return Complex(x - a.x, y - a.y); }
    Complex operator * (const Complex &a) const { return Complex(x * a.x - y * a.y, x * a.y + y * a.x); }
} a[N], b[N];

void fft(Complex *a, int n, int f = 1) { // a 是要 ntt 的序列，n 是长度（2^l），f 是正向运算 (1) 还是逆向运算（-1）
    for (int i = 0, j = 0; i < n; ++i) {
        if (i < j) std::swap(a[i], a[j]);
        for (int l = n >> 1; (j ^= l) < l; l >>= 1) ;
    }
    for (int i = 1; i < n; i <<= 1) {
        Complex w(cos(PI / i), f * sin(PI / i));
        for (int j = 0; j < n; j += (i << 1)) {
            Complex e(1, 0);
            for (int k = 0; k < i; ++k, e = e * w) {
                Complex x = a[j + k], y = e * a[i + j + k];
                a[j + k] = x + y, a[i + j + k] = x - y;
            }
        }
    }
    if (f < 0) for (int i = 0; i < n; ++i) a[i].x /= n;
}
```

### FWT

- $C_k = \sum_{i \oplus j = k} A_i B_j$
- FWT 完后需要先模一遍

```
template<typename T>
void fwt(LL a[], int n, T f) {
    for (int d = 1; d < n; d *= 2)
        for (int i = 0, t = d * 2; i < n; i += t)
            FOR (j, 0, d)
                f(a[i + j], a[i + j + d]);
}

void AND(LL& a, LL& b) { a += b; }
void OR(LL& a, LL& b) { b += a; }
void XOR (LL& a, LL& b) {
    LL x = a, y = b;
    a = (x + y) % MOD;
    b = (x - y + MOD) % MOD;
}
void rAND(LL& a, LL& b) { a -= b; }
void rOR(LL& a, LL& b) { b -= a; }
void rXOR(LL& a, LL& b) {
    static LL INV2 = (MOD + 1) / 2;
    LL x = a, y = b;
    a = (x + y) * INV2 % MOD;
    b = (x - y + MOD) * INV2 % MOD;
}
```

- FWT 子集卷积

```
a[popcount(x)][x] = A[x]
b[popcount(x)][x] = B[x]
fwt(a[i]) fwt(b[i])
c[i + j][x] += a[i][x] * b[j][x]
rfwt(c[i])
ans[x] = c[popcount(x)][x]
```

### 多项式求逆

前置：NTT、取模模板

```cpp
void getinv(int *f) {
    g[0] = fpow(f[0], P - 2);
    for (int d = 2; d < (n << 1); d <<= 1) {
        int l = d << 1;
        for (int i = 0; i < d; ++i) A[i] = f[i];
        ntt(A, l, 1), ntt(g, l, 1);
        for (int i = 0; i < l; ++i) g[i] = g[i] * (2 - (ll)A[i] * g[i] % P + P) % P;
        ntt(g, l, -1);
        for (int i = d; i < l; ++i) g[i] = 0;
    }
}
```

### 分治 fft

```cpp
void cdqfft(int l, int r) { // 注意：l, r 从 0 开始编号，一般调用时取 [0, n - 1]
    if (l == r) return;
    int mid = (l + r) >> 1, L = 1;
    cdqfft(l, mid);
    while (L <= (mid - l + r - l)) L <<= 1;
    std::fill(A, A + L, 0), std::fill(B, B + L, 0);
    for (int i = 0; i < mid - l + 1; ++i) A[i] = f[i + l];
    for (int i = 1; i <= r - l; ++i) B[i] = g[i];
    ntt(A, L, 1), ntt(B, L, 1);
    for (int i = 0; i < L; ++i) A[i] = (ll)A[i] * B[i] % P;
    ntt(A, L, -1);
    for (int i = mid - l + 1; i < r - l + 1; ++i) sadd(f[i + l], A[i]);
    cdqfft(mid + 1, r);
}
```

### 多项式模板大全

来自某位大佬

```cpp
namespace Polynomial {
    using Poly = std::vector<int>;
    constexpr int P(998244353), G(3);
    inline void inc(int &x, int y) { (x += y) >= P ? x -= P : 0; }
    inline int mod(int64_t x) { return x % P; }
    inline int fpow(int x, int k = P - 2) {
        int r = 1;
        for (; k; k >>= 1, x = 1LL * x * x % P)
            if (k & 1) r = 1LL * r * x % P;
        return r;
    }
    template <int N>
    std::array<int, N> getOmega() {
        std::array<int, N> w;
        for (int i = N >> 1, x = fpow(G, (P - 1) / N); i; i >>= 1, x = 1LL * x * x % P) {
            w[i] = 1;
            for (int j = 1; j < i; j++) w[i + j] = 1LL * w[i + j - 1] * x % P;
        }
        return w;
    }
    auto w = getOmega<1 << 18>();
    Poly &operator*=(Poly &a, int b) {
        for (auto &x : a) x = 1LL * x * b % P;
        return a;
    }
    Poly operator*(Poly a, int b) { return a *= b; }
    Poly operator*(int a, Poly b) { return b * a; }
    Poly &operator/=(Poly &a, int b) { return a *= fpow(b); }
    Poly operator/(Poly a, int b) { return a /= b; }
    Poly &operator+=(Poly &a, Poly b) {
        a.resize(std::max(a.size(), b.size()));
        for (int i = 0; i < b.size(); i++) inc(a[i], b[i]);
        return a;
    }
```

```cpp
Poly operator+(Poly a, Poly b) { return a += b; }
Poly &operator-=(Poly &a, Poly b) {
    a.resize(std::max(a.size(), b.size()));
    for (int i = 0; i < b.size(); i++) inc(a[i], P - b[i]);
    return a;
}
Poly operator-(Poly a, Poly b) { return a -= b; }
Poly operator-(Poly a) {
    for (auto &x : a) x ? x = P - x : 0;
    return a;
}
Poly &operator>>=(Poly &a, int x) {
    if (x >= (int)a.size()) {
        a.clear();
    } else {
        a.erase(a.begin(), a.begin() + x);
    }
    return a;
}
Poly &operator<<=(Poly &a, int x) {
    a.insert(a.begin(), x, 0);
    return a;
}
Poly operator>>(Poly a, int x) { return a >>= x; }
Poly operator<<(Poly a, int x) { return a <<= x; }
inline Poly &dotEq(Poly &a, Poly b) {
    assert(a.size() == b.size());
    for (int i = 0; i < a.size(); i++) a[i] = 1LL * a[i] * b[i] % P;
    return a;
}
inline Poly dot(Poly a, Poly b) { return dotEq(a, b); }
void norm(Poly &a) {
    if (!a.empty()) {
        a.resize(1 << std::__lg(a.size() * 2 - 1));
    }
}
void dft(int *a, int n) {
    assert((n & n - 1) == 0);
    for (int k = n >> 1; k; k >>= 1) {
        for (int i = 0; i < n; i += k << 1) {
            for (int j = 0; j < k; j++) {
                int y = a[i + j + k];
                a[i + j + k] = 1LL * (a[i + j] - y + P) * w[k + j] % P;
                inc(a[i + j], y);
            }
        }
    }
}
void idft(int *a, int n) {
    assert((n & n - 1) == 0);
    for (int k = 1; k < n; k <<= 1) {
        for (int i = 0; i < n; i += k << 1) {
            for (int j = 0; j < k; j++) {
                int x = a[i + j], y = 1LL * a[i + j + k] * w[k + j] % P;
                a[i + j + k] = x - y < 0 ? x - y + P : x - y;
                inc(a[i + j], y);
            }
        }
    }
    for (int i = 0, inv = P - (P - 1) / n; i < n; i++) a[i] = 1LL * a[i] * inv % P;
    std::reverse(a + 1, a + n);
}
void dft(Poly &a) { dft(a.data(), a.size()); }
void idft(Poly &a) { idft(a.data(), a.size()); }
// expand dft of length n to 2n
void dftDoubling(int *x, int n) {
    std::copy_n(x, n, x + n);
    idft(x + n, n);
    for (int i = 0; i < n; i++) x[n + i] = 1LL * w[n + i] * x[n + i] % P;
    dft(x + n, n);
}
```

```cpp
    Poly operator*(Poly a, Poly b) {
        int len = a.size() + b.size() - 1;
        if (a.size() <= 8 || b.size() <= 8) {
            Poly c(len);
            for (size_t i = 0; i < a.size(); i++)
                for (size_t j = 0; j < b.size(); j++) c[i + j] = (c[i + j] + 1LL * a[i] * b[j]) % P;
            return c;
        }
        int n = 1 << std::__lg(len - 1) + 1;
        a.resize(n), b.resize(n);
        dft(a), dft(b);
        dotEq(a, b);
        idft(a);
        a.resize(len);
        return a;
    }
    // poly inverse
    // len(a) must be power of 2
    // $O(n \log n)$
    Poly invRec(Poly a) {
        int n = a.size();
        assert((n & n - 1) == 0);
        if (n == 1) return { fpow(a[0]) };
        int m = n >> 1;
        Poly b = invRec(Poly(a.begin(), a.begin() + m)), c = b;
        b.resize(n);
        dft(a), dft(b), dotEq(a, b), idft(a);
        for (int i = 0; i < m; i++) a[i] = 0;
        for (int i = m; i < n; i++) a[i] = P - a[i];
        dft(a), dotEq(a, b), idft(a);
        for (int i = 0; i < m; i++) a[i] = c[i];
        return a;
    }
    // arbitary length
    Poly inverse(Poly a) {
        int n = a.size();
        norm(a);
        a = invRec(a);
        a.resize(n);
        return a;
    }
    // get c where a = b * c + r, (len(a) = n, len(b) = m, len(c) = n - m + 1).
    // $O(n \log n)$
    Poly operator/(Poly a, Poly b) {
        int n = a.size(), m = b.size();
        if (n < m) return { 0 };
        int k = 1 << std::__lg(n - m << 1 | 1);
        std::reverse(a.begin(), a.end());
        std::reverse(b.begin(), b.end());
        a.resize(k), b.resize(k), b = invRec(b);
        a = a * b;
        a.resize(n - m + 1);
        std::reverse(a.begin(), a.end());
        return a;
    }
    // get {c, r}, len(c) = n - m + 1, len(r) = m - 1
    // $O(n \log n)$
    std::pair<Poly, Poly> operator%(Poly a, Poly b) {
        int m = b.size();
        Poly c = a / b;
        b = b * c;
        a.resize(m - 1);
        for (int i = 0; i < m - 1; i++) inc(a[i], P - b[i]);
        return { c, a };
    }
    // $\sqrt a$
    // $O(n \log n)$
    Poly sqrt(Poly a) {
        int raw = a.size();
        int d = 0;
        while (d < raw && !a[d]) d++;
```

```cpp
        if (d == raw) return a;
        if (d & 1) return {};
        norm(a >>= d);
        int len = a.size();
        Poly b(len), binv(1), bsqr{ a[0] }, foo, bar;  // sqrt, sqrt_inv, sqrt_sqr
        // auto sq = SqrtMod::sqrtMod(a[0], P);
        // if (sq.empty()) return {};
        // b[0] = sq[0], binv[0] = fpow(b[0]);
        assert(a[0] == 1);
        b[0] = binv[0] = 1;
        auto shift = [](int x) { return (x & 1 ? x + P : x) >> 1; };  // quick div 2
        for (int m = 1, n = 2; n <= len; m <<= 1, n <<= 1) {
            foo.resize(n), bar = binv;
            for (int i = 0; i < m; i++) {
                foo[i + m] = a[i] + a[i + m] - bsqr[i];
                if (foo[i + m] >= P) foo[i + m] -= P;
                if (foo[i + m] < 0) foo[i + m] += P;
                foo[i] = 0;
            }
            binv.resize(n);
            dft(foo), dft(binv), dotEq(foo, binv), idft(foo);
            for (int i = m; i < n; i++) b[i] = shift(foo[i]);
            // inv
            if (n == len) break;
            for (int i = 0; i < n; i++) foo[i] = b[i];
            bar.resize(n), binv = bar;
            dft(foo), dft(bar), bsqr = dot(foo, foo), idft(bsqr);
            dotEq(foo, bar), idft(foo);
            for (int i = 0; i < m; i++) foo[i] = 0;
            for (int i = m; i < n; i++) foo[i] = P - foo[i];
            dft(foo), dotEq(foo, bar), idft(foo);
            for (int i = m; i < n; i++) binv[i] = foo[i];
        }
        b <<= d / 2;
        b.resize(raw);
        return b;
    }
    // $O(n)$
    Poly deriv(Poly a) {
        for (int i = 0; i + 1 < a.size(); i++) a[i] = (i + 1LL) * a[i + 1] % P;
        a.pop_back();
        return a;
    }
    std::vector<int> inv = { 1, 1 };
    void updateInv(int n) {
        if ((int)inv.size() <= n) {
            int p = inv.size();
            inv.resize(n + 1);
            for (int i = p; i <= n; i++) inv[i] = 1LL * (P - P / i) * inv[P % i] % P;
        }
    }
    // $O(n)$
    Poly integ(Poly a, int c = 0) {
        int n = a.size();
        updateInv(n);
        Poly b(n + 1);
        b[0] = c;
        for (int i = 0; i < n; i++) b[i + 1] = 1LL * inv[i + 1] * a[i] % P;
        return b;
    }
    // $O(n \log n)$
    Poly ln(Poly a) {
        int n = a.size();
        assert(a[0] == 1);
        a = inverse(a) * deriv(a);
        a.resize(n - 1);
        return integ(a);
    }
    // newton
    // $O(n \log n)$, slower than exp2
    Poly expNewton(Poly a) {
```

```cpp
        int n = a.size();
        assert((n & n - 1) == 0);
        assert(a[0] == 0);
        if (n == 1) return { 1 };
        int m = n >> 1;
        Poly b = expNewton(Poly(a.begin(), a.begin() + m)), c;
        b.resize(n), c = ln(b);
        a.resize(n << 1), b.resize(n << 1), c.resize(n << 1);
        dft(a), dft(b), dft(c);
        for (int i = 0; i < n << 1; i++) a[i] = (1LL + P + a[i] - c[i]) * b[i] % P;
        idft(a);
        a.resize(n);
        return a;
    }
    // half-online conv
    // $O(n\log^2n)$
    // $b = e^a, b' = a'b$
    // $(n+1)b_{n+1} = \sum_{i=0}^n a'_ib_{n-i}$
    // $nb_n = \sum_{i=0}^{n-1} a'_ib_{n - 1 - i}$
    Poly exp2(Poly a) {
        if (a.empty()) return {};
        assert(a[0] == 0);
        int n = a.size();
        updateInv(n);
        for (int i = 0; i + 1 < n; i++) {
            a[i] = a[i + 1] * (i + 1LL) % P;
        }
        a.pop_back();
        Poly b(n);
        b[0] = 1;
        for (int m = 1; m < n; m++) {
            int k = m & -m, l = m - k, r = std::min(m + k, n);
            Poly p(a.begin(), a.begin() + (r - l - 1));
            Poly q(b.begin() + l, b.begin() + m);
            p.resize(k * 2), q.resize(k * 2);
            dft(p), dft(q);
            dotEq(p, q);
            idft(p);
            for (int i = m; i < r; i++) inc(b[i], p[i - l - 1]);
            b[m] = 1LL * b[m] * inv[m] % P;
        }
        return b;
    }
    // half-online conv
    // $O(\frac{n\log^2n}{\log\log n})$
    // $nb_n = \sum_{i=0}^{n-1} a'_ib_{n - 1 - i}$
    Poly exp(Poly a) {
        if (a.empty()) return {};
        assert(a[0] == 0);
        int n = a.size();
        updateInv(n);
        for (int i = 0; i + 1 < n; i++) {
            a[i] = a[i + 1] * (i + 1LL) % P;
        }
        a.pop_back();
        Poly b(n);
        b[0] = 1;
        std::vector<Poly> val_a[6], val_b(n);
        for (int m = 1; m < n; m++) {
            int k = 1, d = 0;
            while (!(m / k & 0xf)) k *= 16, d++;
            int l = m & ~(0xf * k), r = std::min(n, m + k);
            if (k == 1) {
                for (int i = m; i < r; i++) {
                    for (int j = l; j < m; j++) {
                        b[i] = (b[i] + 1LL * b[j] * a[i - j - 1]) % P;
                    }
                }
            } else {
                assert(d < 6);
                if (val_a[d].empty()) val_a[d].resize(n);
```

```
                val_b[m] = Poly(b.begin() + (m - k), b.begin() + m);
                val_b[m].resize(k * 2);
                dft(val_b[m]);
                Poly res(k * 2);
                for (; l < m; l += k) {
                    auto &p = val_a[d][m - l - k];
                    if (p.empty()) {
                        p = Poly(a.begin() + (m - l - k), a.begin() + (r - l - 1));
                        p.resize(2 * k);
                        dft(p);
                    }
                    auto &q = val_b[l + k];
                    for (int i = 0; i < k * 2; i++) res[i] = (res[i] + 1LL * p[i] * q[i]) % P;
                }
                idft(res);
                for (int i = m; i < r; i++) inc(b[i], res[i - m + k - 1]);
            }
            b[m] = 1LL * b[m] * inv[m] % P;
        }
        return b;
    }
    Poly power(Poly a, int k) {
        int n = a.size();
        long long d = 0;
        while (d < n && !a[d]) d++;
        if (d == n) return a;
        a >>= d;
        int b = fpow(a[0]);
        norm(a *= b);
        a = exp(ln(a) * k) * fpow(b, P - 1 - k % (P - 1));
        a.resize(n);
        d *= k;
        for (int i = n - 1; i >= d; i--) a[i] = a[i - d];
        d = std::min(d, 1LL * n);
        for (int i = d; i; a[--i] = 0)
            ;
        return a;
    }
    Poly power(Poly a, int k1, int k2) {  // k1 = k % (P - 1), k2 = k % P
        int n = a.size();
        long long d = 0;
        while (d < n && !a[d]) d++;
        if (d == n) return a;
        a >>= d;
        int b = fpow(a[0]);
        norm(a *= b);
        a = exp(ln(a) * k2) * fpow(b, P - 1 - k1 % (P - 1));
        a.resize(n);
        d *= k1;
        for (int i = n - 1; i >= d; i--) a[i] = a[i - d];
        d = std::min(d, 1LL * n);
        for (int i = d; i; a[--i] = 0)
            ;
        return a;
    }
    // [x^n](f / g)
    // $O(m \log m \log n)$
    int divAt(Poly f, Poly g, int64_t n) {
        int len = std::max(f.size(), g.size()), m = 1 << std::__lg(len * 2 - 1);
        f.resize(len), g.resize(len);
        for (; n; n >>= 1) {
            f.resize(m * 2), g.resize(m * 2);
            dft(f), dft(g);
            for (int i = 0; i < m * 2; i++) f[i] = 1LL * f[i] * g[i ^ 1] % P;
            for (int i = 0; i < m; i++) g[i] = 1LL * g[i * 2] * g[i * 2 + 1] % P;
            g.resize(m);
            idft(f), idft(g);
            for (int i = 0, j = n & 1; i < len; i++, j += 2) f[i] = f[j];
            f.resize(len), g.resize(len);
        }
        return f[0];
```

```
    }
    // [x^i](1/q) i = l...r
    Poly invRange(Poly q, int64_t l, int64_t r) {
        assert(l <= r);
        assert(r - l + 1 <= 5e5);
        int len = std::max<int>(q.size(), r - l + 1), m = 1 << std::__lg(len * 2 - 1);

        std::function<Poly(Poly &, int64_t)> cal = [&](Poly &a, int64_t n) -> Poly {
            if (n == 0) {
                Poly res(len);
                int c = 0;
                for (int i = 0; i < m; i++) inc(c, a[i]);
                res.back() = 1LL * m * fpow(c) % P;
                return res;
            }
            dftDoubling(a.data(), m);
            Poly b(m * 2);
            for (int i = 0; i < m; i++) b[i] = 1LL * a[i * 2] * a[i * 2 + 1] % P;
            auto c = cal(b, n >> 1);
            std::fill(b.begin(), b.end(), 0);
            for (int i = 0, o = n & 1 ^ 1; i < len; i++) b[i * 2 ^ o] = c[i];
            dft(b);
            for (int i = 0; i < m * 2; i++) b[i] = 1LL * b[i] * a[i ^ 1] % P;
            idft(b);
            return Poly(b.begin() + len, b.begin() + len * 2);
        };

        q.resize(m * 2);
        dft(q.data(), m);
        q = cal(q, r);
        return Poly(q.end() - (r - l + 1), q.end());
    }
    int divAt2(Poly f, Poly g, int64_t n) {
        g = invRange(g, n - ((int)f.size() - 1), n);
        g = f * g;
        return g[f.size() - 1];
    }
}
```

# 图论

## 最短路

### Dijkstra

```
int dis[N]; bool vis[N];
void dijkstra(int s) {
    static priority_queue<pii, vector<pii>, greater<pii>> q;
    memset(dis, 0x3f, sizeof(dis)); // 最好结合点数清空范围, 否则会很慢
    dis[s] = 0, q.push(pii(dis[s], s));
    while (!q.empty()) {
        int x = q.top().se; q.pop();
        if (vis[x]) continue;
        vis[x] = 1;
        for fec(i, x, y) if (!vis[y] && smin(dis[y], dis[x] + g[i].w)) q.push(pii(dis[y], y)); // for fec 是枚举 x 的每一个孩子, 边存进 i, 孩子存 y
    }
}
```

## 最小生成树

### Kruskal 重构树

```
void kruskal() {
    int tot = 0, cnt = n;
    for (int i = 1; i <= (n << 1); ++i) fa[i] = i;
    sort(e + 1, e + m + 1, cmp);
    for (int i = 1; i <= m; ++i) {
        int u = e[i].u, v = e[i].v;
```

```
        int fx = find(u), fy = find(v);
        if (fx != fy) {
            add(++cnt, fx), add(cnt, fy);
            fa[fx] = cnt, fa[fy] = cnt;
            p[cnt].a = e[i].a;
            ++tot;
        }
        if (tot == n - 1) break;
    }
}
```

### LCA

- 倍增

```
void dfs(int u, int fa) {
    pa[u][0] = fa; dep[u] = dep[fa] + 1;
    FOR (i, 1, SP) pa[u][i] = pa[pa[u][i - 1]][i - 1];
    for (int& v: G[u]) {
        if (v == fa) continue;
        dfs(v, u);
    }
}

int lca(int u, int v) {
    if (dep[u] < dep[v]) swap(u, v);
    int t = dep[u] - dep[v];
    FOR (i, 0, SP) if (t & (1 << i)) u = pa[u][i];
    FORD (i, SP - 1, -1) {
        int uu = pa[u][i], vv = pa[v][i];
        if (uu != vv) { u = uu; v = vv; }
    }
    return u == v ? u : pa[u][0];
}
```

## 网络流

### 最大流 Dinic

```
struct E {
    int to, cp;
    E(int to, int cp): to(to), cp(cp) {}
};

struct Dinic {
    static const int M = 1E5 * 5;
    int m, s, t;
    vector<E> edges;
    vector<int> G[M];
    int d[M];
    int cur[M];

    void init(int n, int s, int t) {
        this->s = s; this->t = t;
        for (int i = 0; i <= n; i++) G[i].clear();
        edges.clear(); m = 0;
    }

    void addedge(int u, int v, int cap) {
        edges.emplace_back(v, cap);
        edges.emplace_back(u, 0);
        G[u].push_back(m++);
        G[v].push_back(m++);
    }

    bool BFS() {
        memset(d, 0, sizeof d);
        queue<int> Q;
        Q.push(s); d[s] = 1;
        while (!Q.empty()) {
```

```cpp
            int x = Q.front(); Q.pop();
            for (int& i: G[x]) {
                E &e = edges[i];
                if (!d[e.to] && e.cp > 0) {
                    d[e.to] = d[x] + 1;
                    Q.push(e.to);
                }
            }
        }
        return d[t];
    }

    int DFS(int u, int cp) {
        if (u == t || !cp) return cp;
        int tmp = cp, f;
        for (int& i = cur[u]; i < G[u].size(); i++) {
            E& e = edges[G[u][i]];
            if (d[u] + 1 == d[e.to]) {
                f = DFS(e.to, min(cp, e.cp));
                e.cp -= f;
                edges[G[u][i] ^ 1].cp += f;
                cp -= f;
                if (!cp) break;
            }
        }
        return tmp - cp;
    }

    int go() {
        int flow = 0;
        while (BFS()) {
            memset(cur, 0, sizeof cur);
            flow += DFS(s, INF);
        }
        return flow;
    }
} DC;
```

### 最大流 ISAP

```cpp
int dis[N], gap[N], cur[N], q[N];
void bfs() {
    int hd = 0, tl = 0;
    q[++tl] = T, ++gap[dis[T] = 1];
    while (hd < tl) {
        int x = q[++hd];
        for fec(i, x, y) if (!dis[y] && !g[i].f) dis[y] = dis[x] + 1, ++gap[dis[y]], q[++tl] = y;
    }
}
int dfs(int x, int a) {
    if (x == T || !a) return a;
    int flow = 0, f;
    for (int &i = cur[x]; i; i = g[i].ne)
        if (dis[x] == dis[g[i].to] + 1 && (f = dfs(g[i].to, std::min(a, g[i].f)))) {
            g[i].f -= f, g[i ^ 1].f += f;
            a -= f, flow += f;
            if (!a) return flow;
        }
    --gap[dis[x]];
    if (!gap[dis[x]]) dis[S] = nod + 1;
    ++gap[++dis[x]];
    return flow;
}
int ISAP() {
    int ans = 0;f
    bfs();
    while (dis[S] <= nod) memcpy(cur, head, sizeof(head)), ans += dfs(S, INF);
    return ans;
}
```

### 费用流

```
struct E {
    int from, to, cp, v;
    E() {}
    E(int f, int t, int cp, int v) : from(f), to(t), cp(cp), v(v) {}
};

struct MCMF {
    int n, m, s, t;
    vector<E> edges;
    vector<int> G[M];
    bool inq[M];
    int d[M], p[M], a[M];

    void init(int _n, int _s, int _t) {
        n = _n; s = _s; t = _t;
        FOR (i, 0, n + 1) G[i].clear();
        edges.clear(); m = 0;
    }

    void addedge(int from, int to, int cap, int cost) {
        edges.emplace_back(from, to, cap, cost);
        edges.emplace_back(to, from, 0, -cost);
        G[from].push_back(m++);
        G[to].push_back(m++);
    }

    bool BellmanFord(int &flow, int &cost) {
        FOR (i, 0, n + 1) d[i] = INF;
        memset(inq, 0, sizeof inq);
        d[s] = 0, a[s] = INF, inq[s] = true;
        queue<int> Q; Q.push(s);
        while (!Q.empty()) {
            int u = Q.front(); Q.pop();
            inq[u] = false;
            for (int& idx: G[u]) {
                E &e = edges[idx];
                if (e.cp && d[e.to] > d[u] + e.v) {
                    d[e.to] = d[u] + e.v;
                    p[e.to] = idx;
                    a[e.to] = min(a[u], e.cp);
                    if (!inq[e.to]) {
                        Q.push(e.to);
                        inq[e.to] = true;
                    }
                }
            }
        }
        if (d[t] == INF) return false;
        flow += a[t];
        cost += a[t] * d[t];
        int u = t;
        while (u != s) {
            edges[p[u]].cp -= a[t];
            edges[p[u] ^ 1].cp += a[t];
            u = edges[p[u]].from;
        }
        return true;
    }

    int go() {
        int flow = 0, cost = 0;
        while (BellmanFord(flow, cost));
        return cost;
    }
} MM;
```

### zkw 费用流

- hkk 版

```cpp
bool spfa() {
    std::fill(dis + 1, dis + n + 1, INF);
    int hd = 0, tl = 0;
    dis[s] = 0, q[++tl] = s, inq[s] = 1;
    while (hd < tl) {
        int x = q[++hd];
        inq[x] = 0;
        for fec(i, x, y) if (g[i].f && smin(dis[y], dis[x] + g[i].w))
            if(!inq[y]) q[++tl] = y;
    }
    return dis[t] < INF;
}
int dfs(int x, int a) {
    vis[x] = 1;
    if (x == t || !a) return a;
    int flow = 0, f;
    for (int &i = cur[x]; i; i = g[i].ne)
        if (!vis[g[i].to] && g[i].f && dis[g[i].to] == dis[x] + g[i].w && (f = dfs(g[i].to, std::min(a, g[i].f)))) {
            g[i].f -= f, g[i ^ 1].f += f;
            ans += f * g[i].w, a -= f, flow += f;
            if (!a) break;
        }
    return flow;
}
int mcmf() {
    int ans = 0;
    while (spfa()) {
        do {
            memcpy(cur, head, sizeof(int) * (n + 1));
            memset(vis, 0, sizeof(bool) * (n + 1));
            ans += dfs(s, INF);
        } while (vis[t]);
    }
    return ans;
}
```

- 模板原版
- 不允许有负权边

```cpp
struct E {
    int to, cp, v;
    E() {}
    E(int to, int cp, int v): to(to), cp(cp), v(v) {}
};

struct MCMF {
    int n, m, s, t, cost, D;
    vector<E> edges;
    vector<int> G[N];
    bool vis[N];

    void init(int _n, int _s, int _t) {
        n = _n; s = _s; t = _t;
        FOR (i, 0, n + 1) G[i].clear();
        edges.clear(); m = 0;
    }

    void addedge(int from, int to, int cap, int cost) {
        edges.emplace_back(to, cap, cost);
        edges.emplace_back(from, 0, -cost);
        G[from].push_back(m++);
        G[to].push_back(m++);
    }

    int aug(int u, int cp) {
        if (u == t) {
            cost += D * cp;
            return cp;
        }
        vis[u] = true;
        int tmp = cp;
```

```cpp
        for (int idx: G[u]) {
            E& e = edges[idx];
            if (e.cp && !e.v && !vis[e.to]) {
                int f = aug(e.to, min(cp, e.cp));
                e.cp -= f;
                edges[idx ^ 1].cp += f;
                cp -= f;
                if (!cp) break;
            }
        }
        return tmp - cp;
    }

    bool modlabel() {
        int d = INF;
        FOR (u, 0, n + 1)
            if (vis[u])
                for (int& idx: G[u]) {
                    E& e = edges[idx];
                    if (e.cp && !vis[e.to]) d = min(d, e.v);
                }
        if (d == INF) return false;
        FOR (u, 0, n + 1)
            if (vis[u])
                for (int& idx: G[u]) {
                    edges[idx].v -= d;
                    edges[idx ^ 1].v += d;
                }
        D += d;
        return true;
    }

    int go(int k) {
        cost = D = 0;
        int flow = 0;
        while (true) {
            memset(vis, 0, sizeof vis);
            int t = aug(s, INF);
            if (!t && !modlabel()) break;
            flow += t;
        }
        return cost;
    }
} MM;
```

**带下界网络流：**

- 无源汇：$u \to v$ 边容量为 $[l, r]$，连容量 $r - l$，虚拟源点到 $v$ 连 $l$，$u$ 到虚拟汇点连 $l$。
- 有源汇：为了让流能循环使用，连 $T \to S$，容量 $\infty$。
- 最大流：跑完可行流后，加 $S' \to S$，$T \to T'$，最大流就是答案（$T \to S$ 的流量自动退回去了，这一部分就是下界部分的流量）。
- 最小流：$T$ 到 $S$ 的那条边的实际流量，减去删掉那条边后 $T$ 到 $S$ 的最大流。
- 网上说可能会减成负的，还要有限地供应 $S$ 之后，再跑一遍 $S$ 到 $T$ 的。
- 费用流：必要的部分（下界以下的）不要钱，剩下的按照最大流。

## 树上路径交

```cpp
int intersection(int x, int y, int xx, int yy) {
    int t[4] = {lca(x, xx), lca(x, yy), lca(y, xx), lca(y, yy)};
    sort(t, t + 4);
    int r = lca(x, y), rr = lca(xx, yy);
    if (dep[t[0]] < min(dep[r], dep[rr]) || dep[t[2]] < max(dep[r], dep[rr]))
        return 0;
    int tt = lca(t[2], t[3]);
    int ret = 1 + dep[t[2]] + dep[t[3]] - dep[tt] * 2;
    return ret;
}
```

### 树上点分治

```cpp
int get_rt(int u) {
    static int q[N], fa[N], sz[N], mx[N];
    int p = 0, cur = -1;
    q[p++] = u; fa[u] = -1;
    while (++cur < p) {
        u = q[cur]; mx[u] = 0; sz[u] = 1;
        for (int& v: G[u])
            if (!vis[v] && v != fa[u]) fa[q[p++] = v] = u;
    }
    FORD (i, p - 1, -1) {
        u = q[i];
        mx[u] = max(mx[u], p - sz[u]);
        if (mx[u] * 2 <= p) return u;
        sz[fa[u]] += sz[u];
        mx[fa[u]] = max(mx[fa[u]], sz[u]);
    }
    assert(0);
}

void dfs(int u) {
    u = get_rt(u);
    vis[u] = true;
    get_dep(u, -1, 0);
    // ...
    for (E& e: G[u]) {
        int v = e.to;
        if (vis[v]) continue;
        // ...
        dfs(v);
    }
}
```

- 动态点分治

```cpp
const int N = 15E4 + 100, INF = 1E9;
struct E {
    int to, d;
};
vector<E> G[N];
int n, Q, w[N];
LL A, ans;

bool vis[N];
int sz[N];

int get_rt(int u) {
    static int q[N], fa[N], sz[N], mx[N];
    int p = 0, cur = -1;
    q[p++] = u; fa[u] = -1;
    while (++cur < p) {
        u = q[cur]; mx[u] = 0; sz[u] = 1;
        for (int& v: G[u])
            if (!vis[v] && v != fa[u]) fa[q[p++] = v] = u;
    }
    FORD (i, p - 1, -1) {
        u = q[i];
        mx[u] = max(mx[u], p - sz[u]);
        if (mx[u] * 2 <= p) return u;
        sz[fa[u]] += sz[u];
        mx[fa[u]] = max(mx[fa[u]], sz[u]);
    }
    assert(0);
}

int dep[N], md[N];
void get_dep(int u, int fa, int d) {
    dep[u] = d; md[u] = 0;
    for (E& e: G[u]) {
        int v = e.to;
        if (vis[v] || v == fa) continue;
```

```cpp
        get_dep(v, u, d + e.d);
        md[u] = max(md[u], md[v] + 1);
    }
}

struct P {
    int w;
    LL s;
};
using VP = vector<P>;
struct R {
    VP *rt, *rt2;
    int dep;
};
VP pool[N << 1], *pit = pool;
vector<R> tr[N];

void go(int u, int fa, VP* rt, VP* rt2) {
    tr[u].push_back({rt, rt2, dep[u]});
    for (E& e: G[u]) {
        int v = e.to;
        if (v == fa || vis[v]) continue;
        go(v, u, rt, rt2);
    }
}

void dfs(int u) {
    u = get_rt(u);
    vis[u] = true;
    get_dep(u, -1, 0);
    VP* rt = pit++; tr[u].push_back({rt, nullptr, 0});
    for (E& e: G[u]) {
        int v = e.to;
        if (vis[v]) continue;
        go(v, u, rt, pit++);
        dfs(v);
    }
}

bool cmp(const P& a, const P& b) { return a.w < b.w; }

LL query(VP& p, int d, int l, int r) {
    l = lower_bound(p.begin(), p.end(), P{l, -1}, cmp) - p.begin();
    r = upper_bound(p.begin(), p.end(), P{r, -1}, cmp) - p.begin() - 1;
    return p[r].s - p[l - 1].s + 1LL * (r - l + 1) * d;
}

int main() {
    cin >> n >> Q >> A;
    FOR (i, 1, n + 1) scanf("%d", &w[i]);
    FOR (_, 1, n) {
        int u, v, d; scanf("%d%d%d", &u, &v, &d);
        G[u].push_back({v, d}); G[v].push_back({u, d});
    }
    dfs(1);
    FOR (i, 1, n + 1)
        for (R& x: tr[i]) {
            x.rt->push_back({w[i], x.dep});
            if (x.rt2) x.rt2->push_back({w[i], x.dep});
        }
    FOR (it, pool, pit) {
        it->push_back({-INF, 0});
        sort(it->begin(), it->end(), cmp);
        FOR (i, 1, it->size())
            (*it)[i].s += (*it)[i - 1].s;
    }
    while (Q--) {
        int u; LL a, b; scanf("%d%lld%lld", &u, &a, &b);
        a = (a + ans) % A; b = (b + ans) % A;
        int l = min(a, b), r = max(a, b);
        ans = 0;
```

```
        for (R& x: tr[u]) {
            ans += query(*(x.rt), x.dep, l, r);
            if (x.rt2) ans -= query(*(x.rt2), x.dep, l, r);
        }
        printf("%lld\n", ans);
    }
}
```

## 树链剖分

- 初始化需要清空 clk
- 使用 hld::predfs(1, 1); hld::dfs(1, 1);
- hkk 版

```
void dfs1(int x, int fa = 0) {
    dep[x] = dep[fa] + 1, f[x] = fa, siz[x] = 1;
    for fec(i, x, y) if (y != fa) dfs1(y, x), siz[x] += siz[y], siz[y] > siz[son[x]] && (son[x] = y);
}
void dfs2(int x, int pa) {
    top[x] = pa, dfn[x] = ++dfc, pre[dfc] = x;
    if (son[x]) dfs2(son[x], pa);
    for fec(i, x, y) if (y != f[x] && y != son[x]) dfs2(y, y);
}
void modify_path(int x, int y, int k) {
    while (top[x] != top[y]) {
        if (dep[top[x]] < dep[top[y]]) std::swap(x, y);
        // do some thing 重链的范围是 dfn[top[x] 到 dfn[x]
        x = f[top[x]];
    }
    if (dep[x] > dep[y]) std::swap(x, y);
    // do some thing 最后一个重链上有用的范围是 dfn[x] 到 dfn[y]
}
int sum_path(int x, int y) {
    int ans = 0;
    while (top[x] != top[y]) {
        if (dep[top[x]] < dep[top[y]]) std::swap(x, y);
        // 统计重链信息 重链的范围是 dfn[top[x] 到 dfn[x]
        x = f[top[x]];
    }
    if (dfn[x] > dfn[y]) std::swap(x, y);
    // ans 加上最后一条重链，最后一个重链上有用的范围是 dfn[x] 到 dfn[y]
    return ans;
}
```

- 模板原版

```
int fa[N], dep[N], idx[N], out[N], ridx[N];
namespace hld {
    int sz[N], son[N], top[N], clk;
    void predfs(int u, int d) {
        dep[u] = d; sz[u] = 1;
        int& maxs = son[u] = -1;
        for (int& v: G[u]) {
            if (v == fa[u]) continue;
            fa[v] = u;
            predfs(v, d + 1);
            sz[u] += sz[v];
            if (maxs == -1 || sz[v] > sz[maxs]) maxs = v;
        }
    }
    void dfs(int u, int tp) {
        top[u] = tp; idx[u] = ++clk; ridx[clk] = u;
        if (son[u] != -1) dfs(son[u], tp);
        for (int& v: G[u])
            if (v != fa[u] && v != son[u]) dfs(v, v);
        out[u] = clk;
    }
    template<typename T>
    int go(int u, int v, T&& f = [](int, int) {}) {
```

```
        int uu = top[u], vv = top[v];
        while (uu != vv) {
            if (dep[uu] < dep[vv]) { swap(uu, vv); swap(u, v); }
            f(idx[uu], idx[u]);
            u = fa[uu]; uu = top[u];
        }
        if (dep[u] < dep[v]) swap(u, v);
        // choose one
        // f(idx[v], idx[u]);
        // if (u != v) f(idx[v] + 1, idx[u]);
        return v;
    }
    int up(int u, int d) {
        while (d) {
            if (dep[u] - dep[top[u]] < d) {
                d -= dep[u] - dep[top[u]];
                u = top[u];
            } else return ridx[idx[u] - d];
            u = fa[u]; --d;
        }
        return u;
    }
    int finds(int u, int rt) { // 找 u 在 rt 的哪个儿子的子树中
        while (top[u] != top[rt]) {
            u = top[u];
            if (fa[u] == rt) return u;
            u = fa[u];
        }
        return ridx[idx[rt] + 1];
    }
}
```

## 二分图匹配

- 最小覆盖数 = 最大匹配数
- 最大独立集 = 顶点数 - 二分图匹配数
- DAG 最小路径覆盖数 = 结点数 - 拆点后二分图最大匹配数

```
struct MaxMatch {
    int n;
    vector<int> G[N];
    int vis[N], left[N], clk;

    void init(int n) {
        this->n = n;
        FOR (i, 0, n + 1) G[i].clear();
        memset(left, -1, sizeof left);
        memset(vis, -1, sizeof vis);
    }

    bool dfs(int u) {
        for (int v: G[u])
            if (vis[v] != clk) {
                vis[v] = clk;
                if (left[v] == -1 || dfs(left[v])) {
                    left[v] = u;
                    return true;
                }
            }
        return false;
    }

    int match() {
        int ret = 0;
        for (clk = 0; clk <= n; ++clk)
            if (dfs(clk)) ++ret;
        return ret;
    }
} MM;
```

- 二分图最大权完美匹配 KM ($O(n^3)$)

```cpp
namespace R {
    const int M = 400 + 5;
    const int INF = 2E9;
    int n;
    int w[M][M], kx[M], ky[M], py[M], vy[M], slk[M], pre[M];

    LL KM() {
        FOR (i, 1, n + 1)
            FOR (j, 1, n + 1)
                kx[i] = max(kx[i], w[i][j]);
        FOR (i, 1, n + 1) {
            fill(vy, vy + n + 1, 0);
            fill(slk, slk + n + 1, INF);
            fill(pre, pre + n + 1, 0);
            int k = 0, p = -1;
            for (py[k = 0] = i; py[k]; k = p) {
                int d = INF;
                vy[k] = 1;
                int x = py[k];
                FOR (j, 1, n + 1)
                    if (!vy[j]) {
                        int t = kx[x] + ky[j] - w[x][j];
                        if (t < slk[j]) { slk[j] = t; pre[j] = k; }
                        if (slk[j] < d) { d = slk[j]; p = j; }
                    }
                FOR (j, 0, n + 1)
                    if (vy[j]) { kx[py[j]] -= d; ky[j] += d; }
                    else slk[j] -= d;
            }
            for (; k; k = pre[k]) py[k] = py[pre[k]];
        }
        LL ans = 0;
        FOR (i, 1, n + 1) ans += kx[i] + ky[i];
        return ans;
    }
}
```

### 虚树

```cpp
void go(vector<int>& V, int& k) {
    int u = V[k]; f[u] = 0;
    dbg(u, k);
    for (auto& e: G[u]) {
        int v = e.to;
        if (v == pa[u][0]) continue;
        while (k + 1 < V.size()) {
            int to = V[k + 1];
            if (in[to] <= out[v]) {
                go(V, ++k);
                if (key[to]) f[u] += w[to];
                else f[u] += min(f[to], (LL)w[to]);
            } else break;
        }
    }
    dbg(u, f[u]);
}
inline bool cmp(int a, int b) { return in[a] < in[b]; }
LL solve(vector<int>& V) {
    static vector<int> a; a.clear();
    for (int& x: V) a.push_back(x);
    sort(a.begin(), a.end(), cmp);
    FOR (i, 1, a.size())
        a.push_back(lca(a[i], a[i - 1]));
    a.push_back(1);
    sort(a.begin(), a.end(), cmp);
    a.erase(unique(a.begin(), a.end()), a.end());
    dbg(a);
    int tmp; go(a, tmp = 0);
    return f[1];
```

```
}
```

### 欧拉路径

```
int S[N << 1], top;
Edge edges[N << 1];
set<int> G[N];

void DFS(int u) {
    S[top++] = u;
    for (int eid: G[u]) {
        int v = edges[eid].get_other(u);
        G[u].erase(eid);
        G[v].erase(eid);
        DFS(v);
        return;
    }
}

void fleury(int start) {
    int u = start;
    top = 0; path.clear();
    S[top++] = u;
    while (top) {
        u = S[--top];
        if (!G[u].empty())
            DFS(u);
        else path.push_back(u);
    }
}
```

### 强连通分量与 2-SAT

```
int n, m;
vector<int> G[N], rG[N], vs;
int used[N], cmp[N];

void add_edge(int from, int to) {
    G[from].push_back(to);
    rG[to].push_back(from);
}

void dfs(int v) {
    used[v] = true;
    for (int u: G[v]) {
        if (!used[u])
            dfs(u);
    }
    vs.push_back(v);
}

void rdfs(int v, int k) {
    used[v] = true;
    cmp[v] = k;
    for (int u: rG[v])
        if (!used[u])
            rdfs(u, k);
}

int scc() {
    memset(used, 0, sizeof(used));
    vs.clear();
    for (int v = 0; v < n; ++v)
        if (!used[v]) dfs(v);
    memset(used, 0, sizeof(used));
    int k = 0;
    for (int i = (int) vs.size() - 1; i >= 0; --i)
        if (!used[vs[i]]) rdfs(vs[i], k++);
    return k;
}
```

```cpp

int main() {
    cin >> n >> m;
    n *= 2;
    for (int i = 0; i < m; ++i) {
        int a, b; cin >> a >> b;
        add_edge(a - 1, (b - 1) ^ 1);
        add_edge(b - 1, (a - 1) ^ 1);
    }
    scc();
    for (int i = 0; i < n; i += 2) {
        if (cmp[i] == cmp[i + 1]) {
            puts("NIE");
            return 0;
        }
    }
    for (int i = 0; i < n; i += 2) {
        if (cmp[i] > cmp[i + 1]) printf("%d\n", i + 1);
        else printf("%d\n", i + 2);
    }
}
```

### 拓扑排序

```cpp
vector<int> toporder(int n) {
    vector<int> orders;
    queue<int> q;
    for (int i = 0; i < n; i++)
        if (!deg[i]) {
            q.push(i);
            orders.push_back(i);
        }
    while (!q.empty()) {
        int u = q.front(); q.pop();
        for (int v: G[u])
            if (!--deg[v]) {
                q.push(v);
                orders.push_back(v);
            }
    }
    return orders;
}
```

### 一般图匹配

带花树。复杂度 $O(n^3)$。

```cpp
int n;
vector<int> G[N];
int fa[N], mt[N], pre[N], mk[N];
int lca_clk, lca_mk[N];
pair<int, int> ce[N];

void connect(int u, int v) {
    mt[u] = v;
    mt[v] = u;
}
int find(int x) { return x == fa[x] ? x : fa[x] = find(fa[x]); }

void flip(int s, int u) {
    if (s == u) return;
    if (mk[u] == 2) {
        int v1 = ce[u].first, v2 = ce[u].second;
        flip(mt[u], v1);
        flip(s, v2);
        connect(v1, v2);
    } else {
        flip(s, pre[mt[u]]);
        connect(pre[mt[u]], mt[u]);
    }
```

```
}

int get_lca(int u, int v) {
    lca_clk++;
    for (u = find(u), v = find(v); ; u = find(pre[u]), v = find(pre[v])) {
        if (u && lca_mk[u] == lca_clk) return u;
        lca_mk[u] = lca_clk;
        if (v && lca_mk[v] == lca_clk) return v;
        lca_mk[v] = lca_clk;
    }
}

void access(int u, int p, const pair<int, int>& c, vector<int>& q) {
    for (u = find(u); u != p; u = find(pre[u])) {
        if (mk[u] == 2) {
            ce[u] = c;
            q.push_back(u);
        }
        fa[find(u)] = find(p);
    }
}

bool aug(int s) {
    fill(mk, mk + n + 1, 0);
    fill(pre, pre + n + 1, 0);
    iota(fa, fa + n + 1, 0);
    vector<int> q = {s};
    mk[s] = 1;
    int t = 0;
    for (int t = 0; t < (int) q.size(); ++t) {
        // q size can be changed
        int u = q[t];
        for (int &v: G[u]) {
            if (find(v) == find(u)) continue;
            if (!mk[v] && !mt[v]) {
                flip(s, u);
                connect(u, v);
                return true;
            } else if (!mk[v]) {
                int w = mt[v];
                mk[v] = 2; mk[w] = 1;
                pre[w] = v; pre[v] = u;
                q.push_back(w);
            } else if (mk[find(v)] == 1) {
                int p = get_lca(u, v);
                access(u, p, {u, v}, q);
                access(v, p, {v, u}, q);
            }
        }
    }
    return false;
}

int match() {
    fill(mt + 1, mt + n + 1, 0);
    lca_clk = 0;
    int ans = 0;
    FOR (i, 1, n + 1)
        if (!mt[i]) ans += aug(i);
    return ans;
}

int main() {
    int m; cin >> n >> m;
    while (m--) {
        int u, v; scanf("%d%d", &u, &v);
        G[u].push_back(v); G[v].push_back(u);
    }
    printf("%d\n", match());
    FOR (i, 1, n + 1) printf("%d%c", mt[i], i == _i - 1 ? '\n' : ' ');
    return 0;
```

```
}
```

## Tarjan

### 割点

- 判断割点
- 注意原图可能不连通

```
int dfn[N], low[N], clk;
void init() { clk = 0; memset(dfn, 0, sizeof dfn); }
void tarjan(int u, int fa) {
    low[u] = dfn[u] = ++clk;
    int cc = fa != -1;
    for (int& v: G[u]) {
        if (v == fa) continue;
        if (!dfn[v]) {
            tarjan(v, u);
            low[u] = min(low[u], low[v]);
            cc += low[v] >= dfn[u];
        } else low[u] = min(low[u], dfn[v]);
    }
    if (cc > 1) // ...
}
```

### 桥

- 注意原图不连通和重边

```
int dfn[N], low[N], clk;
void init() { memset(dfn, 0, sizeof dfn); clk = 0; }
void tarjan(int u, int fa) {
    low[u] = dfn[u] = ++clk;
    int _fst = 0;
    for (E& e: G[u]) {
        int v = e.to; if (v == fa && ++_fst == 1) continue;
        if (!dfn[v]) {
            tarjan(v, u);
            if (low[v] > dfn[u]) // ...
            low[u] = min(low[u], low[v]);
        } else low[u] = min(low[u], dfn[v]);
    }
}
```

### 强连通分量缩点

```
int low[N], dfn[N], clk, B, bl[N];
vector<int> bcc[N];
void init() { B = clk = 0; memset(dfn, 0, sizeof dfn); }
void tarjan(int u) {
    static int st[N], p;
    static bool in[N];
    dfn[u] = low[u] = ++clk;
    st[p++] = u; in[u] = true;
    for (int& v: G[u]) {
        if (!dfn[v]) {
            tarjan(v);
            low[u] = min(low[u], low[v]);
        } else if (in[v]) low[u] = min(low[u], dfn[v]);
    }
    if (dfn[u] == low[u]) {
        while (1) {
            int x = st[--p]; in[x] = false;
            bl[x] = B; bcc[B].push_back(x);
            if (x == u) break;
        }
        ++B;
    }
}
```

### 点双连通分量 / 广义圆方树

- 数组开两倍
- 一条边也被计入点双了（适合拿来建圆方树），可以用点数 <= 边数过滤

```
struct E { int to, nxt; } e[N];
int hd[N], ecnt;
void addedge(int u, int v) {
    e[ecnt] = {v, hd[u]};
    hd[u] = ecnt++;
}
int low[N], dfn[N], clk, B, bno[N];
vector<int> bc[N], be[N];
bool vise[N];
void init() {
    memset(vise, 0, sizeof vise);
    memset(hd, -1, sizeof hd);
    memset(dfn, 0, sizeof dfn);
    memset(bno, -1, sizeof bno);
    B = clk = ecnt = 0;
}

void tarjan(int u, int feid) {
    static int st[N], p;
    static auto add = [&](int x) {
        if (bno[x] != B) { bno[x] = B; bc[B].push_back(x); }
    };
    low[u] = dfn[u] = ++clk;
    for (int i = hd[u]; ~i; i = e[i].nxt) {
        if ((feid ^ i) == 1) continue;
        if (!vise[i]) { st[p++] = i; vise[i] = vise[i ^ 1] = true; }
        int v = e[i].to;
        if (!dfn[v]) {
            tarjan(v, i);
            low[u] = min(low[u], low[v]);
            if (low[v] >= dfn[u]) {
                bc[B].clear(); be[B].clear();
                while (1) {
                    int eid = st[--p];
                    add(e[eid].to); add(e[eid ^ 1].to);
                    be[B].push_back(eid);
                    if ((eid ^ i) <= 1) break;
                }
                ++B;
            }
        } else low[u] = min(low[u], dfn[v]);
    }
}
```

### 圆方树

- 从仙人掌建圆方树
- N 至少边数 × 2

```
vector<int> G[N];
int nn;

struct E { int to, nxt; };
namespace C {
    E e[N * 2];
    int hd[N], ecnt;
    void addedge(int u, int v) {
        e[ecnt] = {v, hd[u]};
        hd[u] = ecnt++;
    }
    int idx[N], clk, fa[N];
    bool ring[N];
    void init() { ecnt = 0; memset(hd, -1, sizeof hd); clk = 0; }
    void dfs(int u, int feid) {
        idx[u] = ++clk;
        for (int i = hd[u]; ~i; i = e[i].nxt) {
```

```
            if ((i ^ feid) == 1) continue;
            int v = e[i].to;
            if (!idx[v]) {
                fa[v] = u; ring[u] = false;
                dfs(v, i);
                if (!ring[u]) { G[u].push_back(v); G[v].push_back(u); }
            } else if (idx[v] < idx[u]) {
                ++nn;
                G[nn].push_back(v); G[v].push_back(nn); // 强行把环的根放在最前面
                for (int x = u; x != v; x = fa[x]) {
                    ring[x] = true;
                    G[nn].push_back(x); G[x].push_back(nn);
                }
                ring[v] = true;
            }
        }
    }
}
```

## 最小树形图

会篡改边。

```
vector<E> edges;
int in[N], id[N], pre[N], vis[N];
// a copy of n is needed
LL zl_tree(int rt, int n) {
    LL ans = 0;
    int v, _n = n;
    while (1) {
        fill(in, in + n, INF);
        for (E &e: edges) {
            if (e.u != e.v && e.w < in[e.v]) {
                pre[e.v] = e.u;
                in[e.v] = e.w;
            }
        }
        FOR (i, 0, n) if (i != rt && in[i] == INF) return -1;
        int tn = 0;
        fill(id, id + _n, -1); fill(vis, vis + _n, -1);
        in[rt] = 0;
        FOR (i, 0, n) {
            ans += in[v = i];
            while (vis[v] != i && id[v] == -1 && v != rt) {
                vis[v] = i; v = pre[v];
            }
            if (v != rt && id[v] == -1) {
                for (int u = pre[v]; u != v; u = pre[u]) id[u] = tn;
                id[v] = tn++;
            }
        }
        if (tn == 0) break;
        FOR (i, 0, n) if (id[i] == -1) id[i] = tn++;
        for (int i = 0; i < (int) edges.size(); ) {
            auto &e = edges[i];
            v = e.v;
            e.u = id[e.u]; e.v = id[e.v];
            if (e.u != e.v) { e.w -= in[v]; i++; }
            else { swap(e, edges.back()); edges.pop_back(); }
        }
        n = tn; rt = id[rt];
    }
    return ans;
}
```

## 差分约束

一个系统 $n$ 个变量和 $m$ 个约束条件组成，每个约束条件形如 $x_j - x_i \le b_k$。可以发现每个约束条件都形如最短路中的三角不等式 $d_u - d_v \le w_{u,v}$。因此连一条边 $(i, j, b_k)$ 建图。

若要使得所有量两两的值最接近，源点到各点的距离初始成 $0$，跑最远路。

若要使得某一变量与其他变量的差尽可能大，则源点到各点距离初始化成 $\infty$，跑最短路。

## 三元环、四元环

### 四元环

考虑这样一个四元环，将答案统计在度数最大的点 $b$ 上。考虑枚举点 $u$，然后枚举与其相邻的点 $v$，然后再枚举所有度数比 $v$ 大的与 $v$ 相邻的点，这些点显然都可能作为 $b$ 点，我们维护一个计数器来计算之前 $b$ 被枚举多少次，答案加上计数器的值，然后计数器加一。

枚举完 $u$ 之后，我们用和枚举时一样的方法来清空计数器就好了。

任何一个点，与其直接相连的度数大于等于它的点最多只有 $\sqrt{2m}$ 个。所以复杂度 $O(m\sqrt{m})$。

```
LL cycle4() {
    LL ans = 0;
    iota(kth, kth + n + 1, 0);
    sort(kth, kth + n, [&](int x, int y) { return deg[x] < deg[y]; });
    FOR (i, 1, n + 1) rk[kth[i]] = i;
    FOR (u, 1, n + 1)
        for (int v: G[u])
            if (rk[v] > rk[u]) key[u].push_back(v);
    FOR (u, 1, n + 1) {
        for (int v: G[u])
            for (int w: key[v])
                if (rk[w] > rk[u]) ans += cnt[w]++;
        for (int v: G[u])
            for (int w: key[v])
                if (rk[w] > rk[u]) --cnt[w];
    }
    return ans;
}
```

### 三元环

将点分成度入小于 $\sqrt{m}$ 和超过 $\sqrt{m}$ 的两类。现求包含第一类点的三元环个数。由于边数较少，直接枚举两条边即可。由于一个点度数不超过 $\sqrt{m}$，所以一条边最多被枚举 $\sqrt{m}$ 次，复杂度 $O(m\sqrt{m})$。再求不包含第一类点的三元环个数，由于这样的点不超过 $\sqrt{m}$ 个，所以复杂度也是 $O(m\sqrt{m})$。

对于每条无向边 $(u,v)$，如果 $d_u < d_v$，那么连有向边 $(u,v)$，否则有向边 $(v,u)$。度数相等的按第二关键字判断。然后枚举每个点 $x$，假设 $x$ 是三元组中度数最小的点，然后暴力往后面枚举两条边找到 $y$，判断 $(x,y)$ 是否有边即可。复杂度也是 $O(m\sqrt{m})$。

```
int cycle3() {
    int ans = 0;
    for (E &e: edges) { deg[e.u]++; deg[e.v]++; }
    for (E &e: edges) {
        if (deg[e.u] < deg[e.v] || (deg[e.u] == deg[e.v] && e.u < e.v))
            G[e.u].push_back(e.v);
        else G[e.v].push_back(e.u);
    }
    FOR (x, 1, n + 1) {
        for (int y: G[x]) p[y] = x;
        for (int y: G[x]) for (int z: G[y]) if (p[z] == x) ans++;
    }
    return ans;
}
```

## 支配树

- `semi[x]` 半必经点（就是 $x$ 的祖先 $z$ 中，能不经过 $z$ 和 $x$ 之间的树上的点而到达 $x$ 的点中深度最小的）
- `idom[x]` 最近必经点（就是深度最大的根到 $x$ 的必经点）

```
vector<int> G[N], rG[N];
vector<int> dt[N];

namespace tl{
    int fa[N], idx[N], clk, ridx[N];
    int c[N], best[N], semi[N], idom[N];
```

```
    void init(int n) {
        clk = 0;
        fill(c, c + n + 1, -1);
        FOR (i, 1, n + 1) dt[i].clear();
        FOR (i, 1, n + 1) semi[i] = best[i] = i;
        fill(idx, idx + n + 1, 0);
    }
    void dfs(int u) {
        idx[u] = ++clk; ridx[clk] = u;
        for (int& v: G[u]) if (!idx[v]) { fa[v] = u; dfs(v); }
    }
    int fix(int x) {
        if (c[x] == -1) return x;
        int &f = c[x], rt = fix(f);
        if (idx[semi[best[x]]] > idx[semi[best[f]]]) best[x] = best[f];
        return f = rt;
    }
    void go(int rt) {
        dfs(rt);
        FORD (i, clk, 1) {
            int x = ridx[i], mn = clk + 1;
            for (int& u: rG[x]) {
                if (!idx[u]) continue;  // 可能不能到达所有点
                fix(u); mn = min(mn, idx[semi[best[u]]]);
            }
            c[x] = fa[x];
            dt[semi[x] = ridx[mn]].push_back(x);
            x = ridx[i - 1];
            for (int& u: dt[x]) {
                fix(u);
                if (semi[best[u]] != x) idom[u] = best[u];
                else idom[u] = x;
            }
            dt[x].clear();
        }

        FOR (i, 2, clk + 1) {
            int u = ridx[i];
            if (idom[u] != semi[u]) idom[u] = idom[idom[u]];
            dt[idom[u]].push_back(u);
        }
    }
}
```

# 计算几何

## 二维几何：点与向量

```
#define y1 yy1
#define nxt(i) ((i + 1) % s.size())
typedef double LD;
const LD PI = 3.14159265358979323846;
const LD eps = 1E-10;
int sgn(LD x) { return fabs(x) < eps ? 0 : (x > 0 ? 1 : -1); }
struct L;
struct P;
typedef P V;
struct P {
    LD x, y;
    explicit P(LD x = 0, LD y = 0): x(x), y(y) {}
    explicit P(const L& l);
};
struct L {
    P s, t;
    L() {}
    L(P s, P t): s(s), t(t) {}
};

P operator + (const P& a, const P& b) { return P(a.x + b.x, a.y + b.y); }
P operator - (const P& a, const P& b) { return P(a.x - b.x, a.y - b.y); }
```

```
P operator * (const P& a, LD k) { return P(a.x * k, a.y * k); }
P operator / (const P& a, LD k) { return P(a.x / k, a.y / k); }
inline bool operator < (const P& a, const P& b) {
    return sgn(a.x - b.x) < 0 || (sgn(a.x - b.x) == 0 && sgn(a.y - b.y) < 0);
}
bool operator == (const P& a, const P& b) { return !sgn(a.x - b.x) && !sgn(a.y - b.y); }
P::P(const L& l) { *this = l.t - l.s; }
ostream &operator << (ostream &os, const P &p) {
    return (os << "(" << p.x << "," << p.y << ")");
}
istream &operator >> (istream &is, P &p) {
    return (is >> p.x >> p.y);
}

LD dist(const P& p) { return sqrt(p.x * p.x + p.y * p.y); }
LD dot(const V& a, const V& b) { return a.x * b.x + a.y * b.y; }
LD det(const V& a, const V& b) { return a.x * b.y - a.y * b.x; }
LD cross(const P& s, const P& t, const P& o = P()) { return det(s - o, t - o); }
// --------------------------------------------
```

### 象限

```
// 象限
int quad(P p) {
    int x = sgn(p.x), y = sgn(p.y);
    if (x > 0 && y >= 0) return 1;
    if (x <= 0 && y > 0) return 2;
    if (x < 0 && y <= 0) return 3;
    if (x >= 0 && y < 0) return 4;
    assert(0);
}

// 仅适用于参照点在所有点一侧的情况
struct cmp_angle {
    P p;
    bool operator () (const P& a, const P& b) {
//        int qa = quad(a - p), qb = quad(b - p);
//        if (qa != qb) return qa < qb;
        int d = sgn(cross(a, b, p));
        if (d) return d > 0;
        return dist(a - p) < dist(b - p);
    }
};
```

### 线

```
// 是否平行
bool parallel(const L& a, const L& b) {
    return !sgn(det(P(a), P(b)));
}
// 直线是否相等
bool l_eq(const L& a, const L& b) {
    return parallel(a, b) && parallel(L(a.s, b.t), L(b.s, a.t));
}
// 逆时针旋转 r 弧度
P rotation(const P& p, const LD& r) { return P(p.x * cos(r) - p.y * sin(r), p.x * sin(r) + p.y * cos(r)); }
P RotateCCW90(const P& p) { return P(-p.y, p.x); }
P RotateCW90(const P& p) { return P(p.y, -p.x); }
// 单位法向量
V normal(const V& v) { return V(-v.y, v.x) / dist(v); }
```

### 点与线

```
// 点在线段上  <= 0 包含端点 < 0 则不包含
bool p_on_seg(const P& p, const L& seg) {
    P a = seg.s, b = seg.t;
    return !sgn(det(p - a, b - a)) && sgn(dot(p - a, p - b)) <= 0;
}
// 点到直线距离
LD dist_to_line(const P& p, const L& l) {
```

```
    return fabs(cross(l.s, l.t, p)) / dist(l);
}
// 点到线段距离
LD dist_to_seg(const P& p, const L& l) {
    if (l.s == l.t) return dist(p - l);
    V vs = p - l.s, vt = p - l.t;
    if (sgn(dot(l, vs)) < 0) return dist(vs);
    else if (sgn(dot(l, vt)) > 0) return dist(vt);
    else return dist_to_line(p, l);
}
```

### 线与线

```
// 求直线交 需要事先保证有界
P l_intersection(const L& a, const L& b) {
    LD s1 = det(P(a), b.s - a.s), s2 = det(P(a), b.t - a.s);
    return (b.s * s2 - b.t * s1) / (s2 - s1);
}
// 向量夹角的弧度
LD angle(const V& a, const V& b) {
    LD r = asin(fabs(det(a, b)) / dist(a) / dist(b));
    if (sgn(dot(a, b)) < 0) r = PI - r;
    return r;
}
// 线段和直线是否有交   1 = 规范, 2 = 不规范
int s_l_cross(const L& seg, const L& line) {
    int d1 = sgn(cross(line.s, line.t, seg.s));
    int d2 = sgn(cross(line.s, line.t, seg.t));
    if ((d1 ^ d2) == -2) return 1; // proper
    if (d1 == 0 || d2 == 0) return 2;
    return 0;
}
// 线段的交   1 = 规范, 2 = 不规范
int s_cross(const L& a, const L& b, P& p) {
    int d1 = sgn(cross(a.t, b.s, a.s)), d2 = sgn(cross(a.t, b.t, a.s));
    int d3 = sgn(cross(b.t, a.s, b.s)), d4 = sgn(cross(b.t, a.t, b.s));
    if ((d1 ^ d2) == -2 && (d3 ^ d4) == -2) { p = l_intersection(a, b); return 1; }
    if (!d1 && p_on_seg(b.s, a)) { p = b.s; return 2; }
    if (!d2 && p_on_seg(b.t, a)) { p = b.t; return 2; }
    if (!d3 && p_on_seg(a.s, b)) { p = a.s; return 2; }
    if (!d4 && p_on_seg(a.t, b)) { p = a.t; return 2; }
    return 0;
}
```

## 多边形

### 面积、凸包

```
typedef vector<P> S;

// 点是否在多边形中 0 = 在外部 1 = 在内部 -1 = 在边界上
int inside(const S& s, const P& p) {
    int cnt = 0;
    FOR (i, 0, s.size()) {
        P a = s[i], b = s[nxt(i)];
        if (p_on_seg(p, L(a, b))) return -1;
        if (sgn(a.y - b.y) <= 0) swap(a, b);
        if (sgn(p.y - a.y) > 0) continue;
        if (sgn(p.y - b.y) <= 0) continue;
        cnt += sgn(cross(b, a, p)) > 0;
    }
    return bool(cnt & 1);
}
// 多边形面积，有向面积可能为负
LD polygon_area(const S& s) {
    LD ret = 0;
    FOR (i, 1, (LL)s.size() - 1)
        ret += cross(s[i], s[i + 1], s[0]);
    return ret / 2;
}
```

```cpp
// 构建凸包 点不可以重复 < 0 边上可以有点， <= 0 则不能
// 会改变输入点的顺序
const int MAX_N = 1000;
S convex_hull(S& s) {
//    assert(s.size() >= 3);
    sort(s.begin(), s.end());
    S ret(MAX_N * 2);
    int sz = 0;
    FOR (i, 0, s.size()) {
        while (sz > 1 && sgn(cross(ret[sz - 1], s[i], ret[sz - 2])) < 0) --sz;
        ret[sz++] = s[i];
    }
    int k = sz;
    FORD (i, (LL)s.size() - 2, -1) {
        while (sz > k && sgn(cross(ret[sz - 1], s[i], ret[sz - 2])) < 0) --sz;
        ret[sz++] = s[i];
    }
    ret.resize(sz - (s.size() > 1));
    return ret;
}

P ComputeCentroid(const vector<P> &p) {
    P c(0, 0);
    LD scale = 6.0 * polygon_area(p);
    for (unsigned i = 0; i < p.size(); i++) {
        unsigned j = (i + 1) % p.size();
        c = c + (p[i] + p[j]) * (p[i].x * p[j].y - p[j].x * p[i].y);
    }
    return c / scale;
}
```

**旋转卡壳**

```cpp
LD rotatingCalipers(vector<P>& qs) {
    int n = qs.size();
    if (n == 2)
        return dist(qs[0] - qs[1]);
    int i = 0, j = 0;
    FOR (k, 0, n) {
        if (!(qs[i] < qs[k])) i = k;
        if (qs[j] < qs[k]) j = k;
    }
    LD res = 0;
    int si = i, sj = j;
    while (i != sj || j != si) {
        res = max(res, dist(qs[i] - qs[j]));
        if (sgn(cross(qs[(i+1)%n] - qs[i], qs[(j+1)%n] - qs[j])) < 0)
            i = (i + 1) % n;
        else j = (j + 1) % n;
    }
    return res;
}

int main() {
    int n;
    while (cin >> n) {
        S v(n);
        FOR (i, 0, n) cin >> v[i].x >> v[i].y;
        convex_hull(v);
        printf("%.0f\n", rotatingCalipers(v));
    }
}
```

**半平面交**

```cpp
struct LV {
    P p, v; LD ang;
    LV() {}
    LV(P s, P t): p(s), v(t - s) { ang = atan2(v.y, v.x); }
};  // 另一种向量表示
```

```cpp

bool operator < (const LV &a, const LV& b) { return a.ang < b.ang; }
bool on_left(const LV& l, const P& p) { return sgn(cross(l.v, p - l.p)) >= 0; }
P l_intersection(const LV& a, const LV& b) {
    P u = a.p - b.p; LD t = cross(b.v, u) / cross(a.v, b.v);
    return a.p + a.v * t;
}

S half_plane_intersection(vector<LV>& L) {
    int n = L.size(), fi, la;
    sort(L.begin(), L.end());
    vector<P> p(n); vector<LV> q(n);
    q[fi = la = 0] = L[0];
    FOR (i, 1, n) {
        while (fi < la && !on_left(L[i], p[la - 1])) la--;
        while (fi < la && !on_left(L[i], p[fi])) fi++;
        q[++la] = L[i];
        if (sgn(cross(q[la].v, q[la - 1].v)) == 0) {
            la--;
            if (on_left(q[la], L[i].p)) q[la] = L[i];
        }
        if (fi < la) p[la - 1] = l_intersection(q[la - 1], q[la]);
    }
    while (fi < la && !on_left(q[fi], p[la - 1])) la--;
    if (la - fi <= 1) return vector<P>();
    p[la] = l_intersection(q[la], q[fi]);
    return vector<P>(p.begin() + fi, p.begin() + la + 1);
}

S convex_intersection(const vector<P> &v1, const vector<P> &v2) {
    vector<LV> h; int n = v1.size(), m = v2.size();
    FOR (i, 0, n) h.push_back(LV(v1[i], v1[(i + 1) % n]));
    FOR (i, 0, m) h.push_back(LV(v2[i], v2[(i + 1) % m]));
    return half_plane_intersection(h);
}
```

### 圆

```cpp
struct C {
    P p; LD r;
    C(LD x = 0, LD y = 0, LD r = 0): p(x, y), r(r) {}
    C(P p, LD r): p(p), r(r) {}
};
```

### 三点求圆心

```cpp
P compute_circle_center(P a, P b, P c) {
    b = (a + b) / 2;
    c = (a + c) / 2;
    return l_intersection({b, b + RotateCW90(a - b)}, {c , c + RotateCW90(a - c)});
}
```

### 圆线交点、圆圆交点

- 圆和线的交点关于圆心是顺时针的

```cpp
vector<P> c_l_intersection(const L& l, const C& c) {
    vector<P> ret;
    P b(l), a = l.s - c.p;
    LD x = dot(b, b), y = dot(a, b), z = dot(a, a) - c.r * c.r;
    LD D = y * y - x * z;
    if (sgn(D) < 0) return ret;
    ret.push_back(c.p + a + b * (-y + sqrt(D + eps)) / x);
    if (sgn(D) > 0) ret.push_back(c.p + a + b * (-y - sqrt(D)) / x);
    return ret;
}

vector<P> c_c_intersection(C a, C b) {
    vector<P> ret;
    LD d = dist(a.p - b.p);
```

```
    if (sgn(d) == 0 || sgn(d - (a.r + b.r)) > 0 || sgn(d + min(a.r, b.r) - max(a.r, b.r)) < 0)
        return ret;
    LD x = (d * d - b.r * b.r + a.r * a.r) / (2 * d);
    LD y = sqrt(a.r * a.r - x * x);
    P v = (b.p - a.p) / d;
    ret.push_back(a.p + v * x + RotateCCW90(v) * y);
    if (sgn(y) > 0) ret.push_back(a.p + v * x - RotateCCW90(v) * y);
    return ret;
}
```

### 圆圆位置关系

```
// 1: 内含 2: 内切 3: 相交 4: 外切 5: 相离
int c_c_relation(const C& a, const C& v) {
    LD d = dist(a.p - v.p);
    if (sgn(d - a.r - v.r) > 0) return 5;
    if (sgn(d - a.r - v.r) == 0) return 4;
    LD l = fabs(a.r - v.r);
    if (sgn(d - l) > 0) return 3;
    if (sgn(d - l) == 0) return 2;
    if (sgn(d - l) < 0) return 1;
}
```

### 圆与多边形交

- HDU 5130
- 注意顺时针逆时针（可能要取绝对值）

```
LD sector_area(const P& a, const P& b, LD r) {
    LD th = atan2(a.y, a.x) - atan2(b.y, b.x);
    while (th <= 0) th += 2 * PI;
    while (th > 2 * PI) th -= 2 * PI;
    th = min(th, 2 * PI - th);
    return r * r * th / 2;
}

LD c_tri_area(P a, P b, P center, LD r) {
    a = a - center; b = b - center;
    int ina = sgn(dist(a) - r) < 0, inb = sgn(dist(b) - r) < 0;
    // dbg(a, b, ina, inb);
    if (ina && inb) {
        return fabs(cross(a, b)) / 2;
    } else {
        auto p = c_l_intersection(L(a, b), C(0, 0, r));
        if (ina ^ inb) {
            auto cr = p_on_seg(p[0], L(a, b)) ? p[0] : p[1];
            if (ina) return sector_area(b, cr, r) + fabs(cross(a, cr)) / 2;
            else return sector_area(a, cr, r) + fabs(cross(b, cr)) / 2;
        } else {
            if ((int) p.size() == 2 && p_on_seg(p[0], L(a, b))) {
                if (dist(p[0] - a) > dist(p[1] - a)) swap(p[0], p[1]);
                return sector_area(a, p[0], r) + sector_area(p[1], b, r)
                    + fabs(cross(p[0], p[1])) / 2;
            } else return sector_area(a, b, r);
        }
    }
}

typedef vector<P> S;
LD c_poly_area(S poly, const C& c) {
    LD ret = 0; int n = poly.size();
    FOR (i, 0, n) {
        int t = sgn(cross(poly[i] - c.p, poly[(i + 1) % n] - c.p));
        if (t) ret += t * c_tri_area(poly[i], poly[(i + 1) % n], c.p, c.r);
    }
    return ret;
}
```

**圆的离散化、面积并**

SPOJ: CIRU, EOJ: 284

- 版本 1：复杂度 $O(n^3 \log n)$。虽然常数小，但还是难以接受。
- 优点？想不出来。
- 原理上是用竖线进行切分，然后对每一个切片分别计算。
- 扫描线部分可以魔改，求各种东西。

```
inline LD rt(LD x) { return sgn(x) == 0 ? 0 : sqrt(x); }
inline LD sq(LD x) { return x * x; }

// 圆弧
// 如果按照 x 离散化，圆弧是 " 横着的"
// 记录圆弧的左端点、右端点、中点的坐标，和圆弧所在的圆
// 调用构造要保证 c.x - x.r <= xl < xr <= c.y + x.r
// t = 1 下圆弧 t = -1 上圆弧
struct CV {
    LD yl, yr, ym; C o; int type;
    CV() {}
    CV(LD yl, LD yr, LD ym, C c, int t)
        : yl(yl), yr(yr), ym(ym), type(t), o(c) {}
};

// 辅助函数 求圆上纵坐标
pair<LD, LD> c_point_eval(const C& c, LD x) {
    LD d = fabs(c.p.x - x), h = rt(sq(c.r) - sq(d));
    return {c.p.y - h, c.p.y + h};
}
// 构造上下圆弧
pair<CV, CV> pairwise_curves(const C& c, LD xl, LD xr) {
    LD yl1, yl2, yr1, yr2, ym1, ym2;
    tie(yl1, yl2) = c_point_eval(c, xl);
    tie(ym1, ym2) = c_point_eval(c, (xl + xr) / 2);
    tie(yr1, yr2) = c_point_eval(c, xr);
    return {CV(yl1, yr1, ym1, c, 1), CV(yl2, yr2, ym2, c, -1)};
}

// 离散化之后同一切片内的圆弧应该是不相交的
bool operator < (const CV& a, const CV& b) { return a.ym < b.ym; }
// 计算圆弧和连接圆弧端点的线段构成的封闭图形的面积
LD cv_area(const CV& v, LD xl, LD xr) {
    LD l = rt(sq(xr - xl) + sq(v.yr - v.yl));
    LD d = rt(sq(v.o.r) - sq(l / 2));
    LD ang = atan(l / d / 2);
    return ang * sq(v.o.r) - d * l / 2;
}

LD circle_union(const vector<C>& cs) {
    int n = cs.size();
    vector<LD> xs;
    FOR (i, 0, n) {
        xs.push_back(cs[i].p.x - cs[i].r);
        xs.push_back(cs[i].p.x);
        xs.push_back(cs[i].p.x + cs[i].r);
        FOR (j, i + 1, n) {
            auto pts = c_c_intersection(cs[i], cs[j]);
            for (auto& p: pts) xs.push_back(p.x);
        }
    }
    sort(xs.begin(), xs.end());
    xs.erase(unique(xs.begin(), xs.end(), [](LD x, LD y) { return sgn(x - y) == 0; }), xs.end());
    LD ans = 0;
    FOR (i, 0, (int) xs.size() - 1) {
        LD xl = xs[i], xr = xs[i + 1];
        vector<CV> intv;
        FOR (k, 0, n) {
            auto& c = cs[k];
            if (sgn(c.p.x - c.r - xl) <= 0 && sgn(c.p.x + c.r - xr) >= 0) {
                auto t = pairwise_curves(c, xl, xr);
                intv.push_back(t.first); intv.push_back(t.second);
```

```
            }
        }
        sort(intv.begin(), intv.end());

        vector<LD> areas(intv.size());
        FOR (i, 0, intv.size()) areas[i] = cv_area(intv[i], xl, xr);

        int cc = 0;
        FOR (i, 0, intv.size()) {
            if (cc > 0) {
                ans += (intv[i].yl - intv[i - 1].yl + intv[i].yr - intv[i - 1].yr) * (xr - xl) / 2;
                ans += intv[i - 1].type * areas[i - 1];
                ans -= intv[i].type * areas[i];
            }
            cc += intv[i].type;
        }
    }
    return ans;
}
```

- 版本 2：复杂度 $O(n^2 \log n)$。
- 原理是：认为所求部分是一个奇怪的多边形 + 若干弓形。然后对于每个圆分别求贡献的弓形，并累加多边形有向面积。
- 同样可以魔改扫描线的部分，用于求周长、至少覆盖 $k$ 次等等。
- 内含、内切、同一个圆的情况，通常需要特殊处理。
- 下面的代码是 $k$ 圆覆盖。

```
inline LD angle(const P& p) { return atan2(p.y, p.x); }

// 圆弧上的点
// p 是相对于圆心的坐标
// a 是在圆上的 atan2 [-PI, PI]
struct CP {
    P p; LD a; int t;
    CP() {}
    CP(P p, LD a, int t): p(p), a(a), t(t) {}
};
bool operator < (const CP& u, const CP& v) { return u.a < v.a; }
LD cv_area(LD r, const CP& q1, const CP& q2) {
    return (r * r * (q2.a - q1.a) - cross(q1.p, q2.p)) / 2;
}

LD ans[N];
void circle_union(const vector<C>& cs) {
    int n = cs.size();
    FOR (i, 0, n) {
        // 有相同的圆的话只考虑第一次出现
        bool ok = true;
        FOR (j, 0, i)
            if (sgn(cs[i].r - cs[j].r) == 0 && cs[i].p == cs[j].p) {
                ok = false;
                break;
            }
        if (!ok) continue;
        auto& c = cs[i];
        vector<CP> ev;
        int belong_to = 0;
        P bound = c.p + P(-c.r, 0);
        ev.emplace_back(bound, -PI, 0);
        ev.emplace_back(bound, PI, 0);
        FOR (j, 0, n) {
            if (i == j) continue;
            if (c_c_relation(c, cs[j]) <= 2) {
                if (sgn(cs[j].r - c.r) >= 0) // 完全被另一个圆包含，等于说叠了一层
                    belong_to++;
                continue;
            }
            auto its = c_c_intersection(c, cs[j]);
            if (its.size() == 2) {
                P p = its[1] - c.p, q = its[0] - c.p;
                LD a = angle(p), b = angle(q);
                if (sgn(a - b) > 0) {
```

```
                    ev.emplace_back(p, a, 1);
                    ev.emplace_back(bound, PI, -1);
                    ev.emplace_back(bound, -PI, 1);
                    ev.emplace_back(q, b, -1);
                } else {
                    ev.emplace_back(p, a, 1);
                    ev.emplace_back(q, b, -1);
                }
            }
        }
        sort(ev.begin(), ev.end());
        int cc = ev[0].t;
        FOR (j, 1, ev.size()) {
            int t = cc + belong_to;
            ans[t] += cross(ev[j - 1].p + c.p, ev[j].p + c.p) / 2;
            ans[t] += cv_area(c.r, ev[j - 1], ev[j]);
            cc += ev[j].t;
        }
    }
}
```

#### 最小圆覆盖

- 随机增量。期望复杂度 $O(n)$。

```
P compute_circle_center(P a, P b) { return (a + b) / 2; }
bool p_in_circle(const P& p, const C& c) {
    return sgn(dist(p - c.p) - c.r) <= 0;
}
C min_circle_cover(const vector<P> &in) {
    vector<P> a(in.begin(), in.end());
    dbg(a.size());
    random_shuffle(a.begin(), a.end());
    P c = a[0]; LD r = 0; int n = a.size();
    FOR (i, 1, n) if (!p_in_circle(a[i], {c, r})) {
        c = a[i]; r = 0;
        FOR (j, 0, i) if (!p_in_circle(a[j], {c, r})) {
            c = compute_circle_center(a[i], a[j]);
            r = dist(a[j] - c);
            FOR (k, 0, j) if (!p_in_circle(a[k], {c, r})) {
                c = compute_circle_center(a[i], a[j], a[k]);
                r = dist(a[k] - c);
            }
        }
    }
    return {c, r};
}
```

#### 圆的反演

```
C inv(C c, const P& o) {
    LD d = dist(c.p - o);
    assert(sgn(d) != 0);
    LD a = 1 / (d - c.r);
    LD b = 1 / (d + c.r);
    c.r = (a - b) / 2 * R2;
    c.p = o + (c.p - o) * ((a + b) * R2 / 2 / d);
    return c;
}
```

### 三维计算几何

```
struct P;
struct L;
typedef P V;

struct P {
    LD x, y, z;
    explicit P(LD x = 0, LD y = 0, LD z = 0): x(x), y(y), z(z) {}
    explicit P(const L& l);
```

```cpp
};

struct L {
    P s, t;
    L() {}
    L(P s, P t): s(s), t(t) {}
};

struct F {
    P a, b, c;
    F() {}
    F(P a, P b, P c): a(a), b(b), c(c) {}
};

P operator + (const P& a, const P& b) { return P(a.x + b.x, a.y + b.y, a.z + b.z); }
P operator - (const P& a, const P& b) { return P(a.x - b.x, a.y - b.y, a.z - b.z); }
P operator * (const P& a, LD k) { return P(a.x * k, a.y * k, a.z * k); }
P operator / (const P& a, LD k) { return P(a.x / k, a.y / k, a.z / k); }
inline int operator < (const P& a, const P& b) {
    return sgn(a.x - b.x) < 0 || (sgn(a.x - b.x) == 0 && (sgn(a.y - b.y) < 0 ||
                                  (sgn(a.y - b.y) == 0 && sgn(a.z - b.z) < 0)));
}
bool operator == (const P& a, const P& b) { return !sgn(a.x - b.x) && !sgn(a.y - b.y) && !sgn(a.z - b.z); }
P::P(const L& l) { *this = l.t - l.s; }
ostream &operator << (ostream &os, const P &p) {
    return (os << "(" << p.x << "," << p.y << "," << p.z << ")");
}
istream &operator >> (istream &is, P &p) {
    return (is >> p.x >> p.y >> p.z);
}

// --------------------------------------------
LD dist2(const P& p) { return p.x * p.x + p.y * p.y + p.z * p.z; }
LD dist(const P& p) { return sqrt(dist2(p)); }
LD dot(const V& a, const V& b) { return a.x * b.x + a.y * b.y + a.z * b.z; }
P cross(const P& v, const P& w) {
    return P(v.y * w.z - v.z * w.y, v.z * w.x - v.x * w.z, v.x * w.y - v.y * w.x);
}
LD mix(const V& a, const V& b, const V& c) { return dot(a, cross(b, c)); }
```

### 旋转

```cpp
// 逆时针旋转 r 弧度
// axis = 0 绕 x 轴
// axis = 1 绕 y 轴
// axis = 2 绕 z 轴
P rotation(const P& p, const LD& r, int axis = 0) {
    if (axis == 0)
        return P(p.x, p.y * cos(r) - p.z * sin(r), p.y * sin(r) + p.z * cos(r));
    else if (axis == 1)
        return P(p.z * cos(r) - p.x * sin(r), p.y, p.z * sin(r) + p.x * cos(r));
    else if (axis == 2)
        return P(p.x * cos(r) - p.y * sin(r), p.x * sin(r) + p.y * cos(r), p.z);
}
// n 是单位向量 表示旋转轴
// 模板是顺时针的
P rotation(const P& p, const LD& r, const P& n) {
    LD c = cos(r), s = sin(r), x = n.x, y = n.y, z = n.z;
    // dbg(c, s);
    return P((x * x * (1 - c) + c) * p.x + (x * y * (1 - c) + z * s) * p.y + (x * z * (1 - c) - y * s) * p.z,
             (x * y * (1 - c) - z * s) * p.x + (y * y * (1 - c) + c) * p.y + (y * z * (1 - c) + x * s) * p.z,
             (x * z * (1 - c) + y * s) * p.x + (y * z * (1 - c) - x * s) * p.y + (z * z * (1 - c) + c) * p.z);
}
```

### 线、面

函数相互依赖，所以交织在一起了。

```cpp
// 点在线段上  <= 0 包含端点 < 0 则不包含
bool p_on_seg(const P& p, const L& seg) {
    P a = seg.s, b = seg.t;
```

```
    return !sgn(dist2(cross(p - a, b - a))) && sgn(dot(p - a, p - b)) <= 0;
}
// 点到直线距离
LD dist_to_line(const P& p, const L& l) {
    return dist(cross(l.s - p, l.t - p)) / dist(l);
}
// 点到线段距离
LD dist_to_seg(const P& p, const L& l) {
    if (l.s == l.t) return dist(p - l.s);
    V vs = p - l.s, vt = p - l.t;
    if (sgn(dot(l, vs)) < 0) return dist(vs);
    else if (sgn(dot(l, vt)) > 0) return dist(vt);
    else return dist_to_line(p, l);
}

P norm(const F& f) { return cross(f.a - f.b, f.b - f.c); }
int p_on_plane(const F& f, const P& p) { return sgn(dot(norm(f), p - f.a)) == 0; }

// 判两点在线段异侧 点在线段上返回 0 不共面无意义
int opposite_side(const P& u, const P& v, const L& l) {
    return sgn(dot(cross(P(l), u - l.s), cross(P(l), v - l.s))) < 0;
}

bool parallel(const L& a, const L& b) { return !sgn(dist2(cross(P(a), P(b)))); }
// 线段相交
int s_intersect(const L& u, const L& v) {
    return p_on_plane(F(u.s, u.t, v.s), v.t) &&
           opposite_side(u.s, u.t, v) &&
           opposite_side(v.s, v.t, u);
}
```

**凸包**

增量法。先将所有的点打乱顺序，然后选择四个不共面的点组成一个四面体，如果找不到说明凸包不存在。然后遍历剩余的点，不断更新凸包。对遍历到的点做如下处理。

1. 如果点在凸包内，则不更新。
2. 如果点在凸包外，那么找到所有原凸包上所有分隔了对于这个点可见面和不可见面的边，以这样的边的两个点和新的点创建新的面加入凸包中。

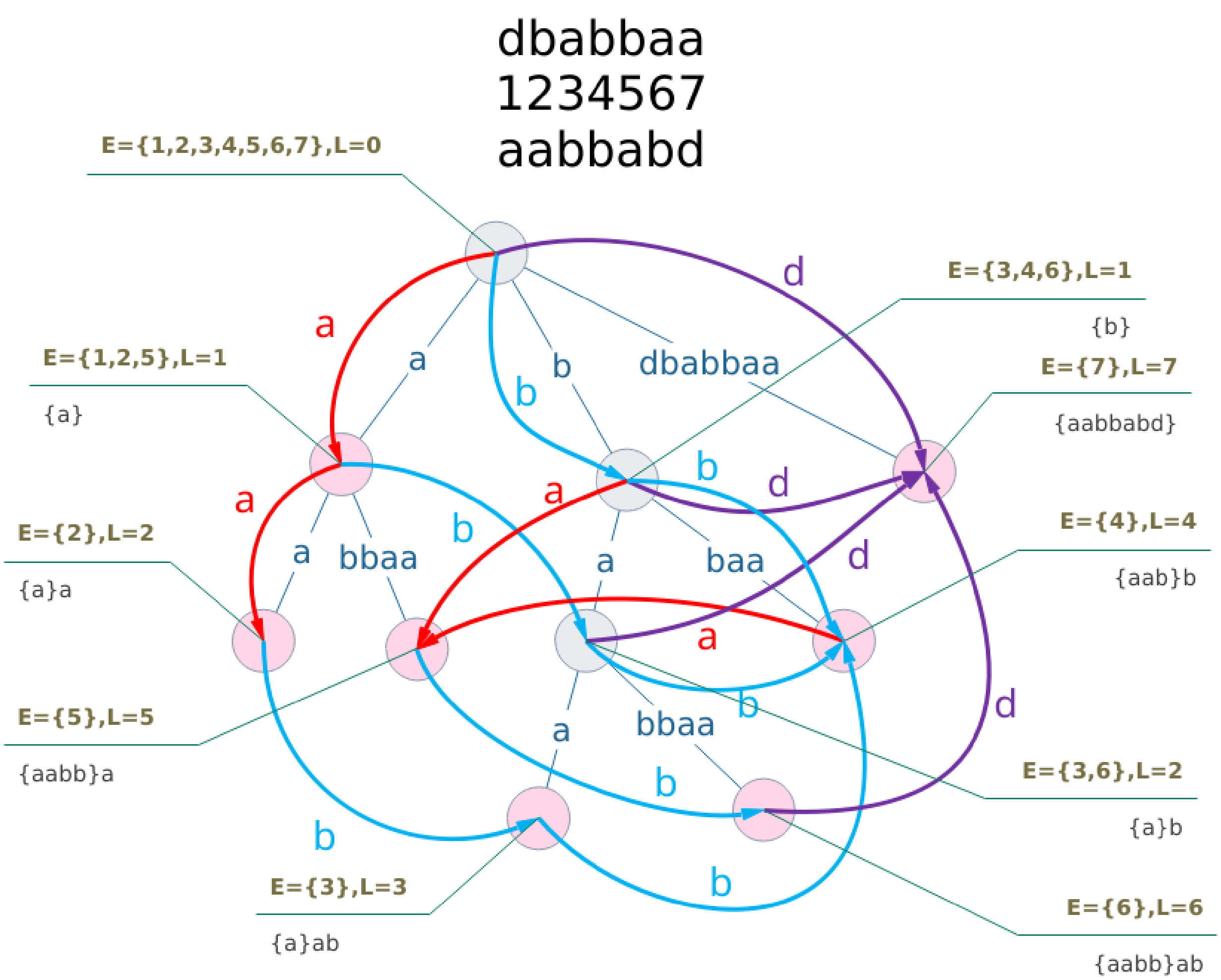
```
struct FT {
    int a, b, c;
    FT() { }
    FT(int a, int b, int c) : a(a), b(b), c(c) { }
};

bool p_on_line(const P& p, const L& l) {
    return !sgn(dist2(cross(p - l.s, P(l))));
}

vector<F> convex_hull(vector<P> &p) {
    sort(p.begin(), p.end());
    p.erase(unique(p.begin(), p.end()), p.end());
    random_shuffle(p.begin(), p.end());
    vector<FT> face;
    FOR (i, 2, p.size()) {
        if (p_on_line(p[i], L(p[0], p[1]))) continue;
        swap(p[i], p[2]);
        FOR (j, i + 1, p.size())
            if (sgn(mix(p[1] - p[0], p[2] - p[1], p[j] - p[0]))) {
                swap(p[j], p[3]);
                face.emplace_back(0, 1, 2);
                face.emplace_back(0, 2, 1);
                goto found;
            }
    }
found:
    vector<vector<int>> mk(p.size(), vector<int>(p.size()));
    FOR (v, 3, p.size()) {
        vector<FT> tmp;
```

```
        FOR (i, 0, face.size()) {
            int a = face[i].a, b = face[i].b, c = face[i].c;
            if (sgn(mix(p[a] - p[v], p[b] - p[v], p[c] - p[v])) < 0) {
                mk[a][b] = mk[b][a] = v;
                mk[b][c] = mk[c][b] = v;
                mk[c][a] = mk[a][c] = v;
            } else tmp.push_back(face[i]);
        }
        face = tmp;
        FOR (i, 0, tmp.size()) {
            int a = face[i].a, b = face[i].b, c = face[i].c;
            if (mk[a][b] == v) face.emplace_back(b, a, v);
            if (mk[b][c] == v) face.emplace_back(c, b, v);
            if (mk[c][a] == v) face.emplace_back(a, c, v);
        }
    }
    vector<F> out;
    FOR (i, 0, face.size())
        out.emplace_back(p[face[i].a], p[face[i].b], p[face[i].c]);
    return out;
}
```

## 字符串

### 后缀自动机

#### 一般的后缀自动机



- 广义后缀自动机如果直接使用以下代码的话会产生一些冗余状态（置 last 为 1），所以要用拓扑排序。用 len 基数排序不能。
- 字符集大的话要使用 `map`。
- 树上 dp 时注意边界（root 和 null）。
- rsort 中的数组 a 是拓扑序 [1, sz)

```cpp
// 注意 N 应该是字符串长度的 2 倍!
int nod = 1, lst = 1;
struct Node { int c[26], len, fa; } t[N];
int tax[N], q[N], sz[N]; // tax 用来对 len 计数的数组, q 排序后的队列, sz 是 parent 树的子树大小
void extend(int x) {
    int p = lst, np = ++nod;
    t[np].len = t[p].len + 1;
    for (; p && !t[p].c[x]; p = t[p].fa) t[p].c[x] = np;
    if (!p) t[np].fa = 1;
    else {
        int q = t[p].c[x];
        if (t[q].len == t[p].len + 1) t[np].fa = q;
        else {
            int nq = ++nod;
            t[nq] = t[q], t[nq].len = t[p].len + 1, t[q].fa = t[np].fa = nq;
            for (; p && t[p].c[x] == q; p = t[p].fa) t[p].c[x] = nq;
        }
    }
    lst = np;
}
void resort() { // 对节点重新排序, 使得在 SAM 和 parent 树上都是拓扑序
    for (int i = 1; i <= nod; ++i) ++tax[t[i].len];
    for (int i = 1; i <= n; ++i) tax[i] += tax[i - 1];
    for (int i = 1; i <= nod; ++i) q[tax[t[i].len]--] = i;
    for (int i = nod; i; --i) sz[t[q[i]].fa] += sz[q[i]];
}
```

**真·广义后缀自动机**

```cpp
// 注意 N 应该是字符串长度的 2 倍!
int nod = 1;
struct Node { int c[26], len, fa; } t[N];
int tax[N], q[N], sz[N]; // tax 用来对 len 计数的数组, q 排序后的队列, sz 是 parent 树的子树大小
int extend(int p, int x) {
    if (t[p].c[x]) {
        int q = t[p].c[x];
        if (t[q].len == t[p].len + 1) return q;
        int nq = ++nod;
        t[nq] = t[q], t[nq].len = t[p].len + 1, t[q].fa = nq;
        for (; t[p].c[x] == q; p = t[p].fa) t[p].c[x] = nq;
        return nq;
    }
    int np = ++nod;
    t[np].len = t[p].len + 1;
    for (; p && !t[p].c[x]; p = t[p].fa) t[p].c[x] = np;
    if (!p) t[np].fa = 1;
    else {
        int q = t[p].c[x];
        if (t[q].len == t[p].len + 1) t[np].fa = q;
        else {
            int nq = ++nod;
            t[nq] = t[q], t[nq].len = t[p].len + 1, t[q].fa = t[np].fa = nq;
            for (; p && t[p].c[x] == q; p = t[p].fa) t[p].c[x] = nq;
        }
    }
    return np;
}
void resort() {
    for (int i = 1; i <= nod; ++i) ++tax[t[i].len];
    for (int i = 1; i <= n; ++i) tax[i] += tax[i - 1];
    for (int i = 1; i <= nod; ++i) q[tax[t[i].len]--] = i;
    for (int i = nod; i; --i) sz[t[q[i]].fa] += sz[q[i]];
}
```

以下为原模板的内容：

- 按字典序建立后缀树注意逆序插入
- rsort2 里的 a 不是拓扑序，需要拓扑序就去树上做

```
void ins(int ch, int pp) {
    int p = last, np = last = sz++;
    len[np] = len[p] + 1; one[np] = pos[np] = pp;
    for (; p && !t[p][ch]; p = fa[p]) t[p][ch] = np;
    if (!p) { fa[np] = 1; return; }
    int q = t[p][ch];
    if (len[q] == len[p] + 1) fa[np] = q;
    else {
        int nq = sz++; len[nq] = len[p] + 1; one[nq] = one[q];
        memcpy(t[nq], t[q], sizeof t[0]);
        fa[nq] = fa[q];
        fa[q] = fa[np] = nq;
        for (; p && t[p][ch] == q; p = fa[p]) t[p][ch] = nq;
    }
}

int up[M], c[256] = {2}, a[M];
void rsort2() {
    FOR (i, 1, 256) c[i] = 0;
    FOR (i, 2, sz) up[i] = s[one[i] + len[fa[i]]];
    FOR (i, 2, sz) c[up[i]]++;
    FOR (i, 1, 256) c[i] += c[i - 1];
    FOR (i, 2, sz) a[--c[up[i]]] = i;
    FOR (i, 2, sz) G[fa[a[i]]].push_back(a[i]);
}
```

- 广义后缀自动机建后缀树，必须反向插入

```
int t[M][26], len[M] = {0}, fa[M], sz = 2, last = 1;
char* one[M];
void ins(int ch, char* pp) {
    int p = last, np = 0, nq = 0, q = -1;
    if (!t[p][ch]) {
        np = sz++; one[np] = pp;
        len[np] = len[p] + 1;
        for (; p && !t[p][ch]; p = fa[p]) t[p][ch] = np;
    }
    if (!p) fa[np] = 1;
    else {
        q = t[p][ch];
        if (len[p] + 1 == len[q]) fa[np] = q;
        else {
            nq = sz++; len[nq] = len[p] + 1; one[nq] = one[q];
            memcpy(t[nq], t[q], sizeof t[0]);
            fa[nq] = fa[q];
            fa[np] = fa[q] = nq;
            for (; t[p][ch] == q; p = fa[p]) t[p][ch] = nq;
        }
    }
    last = np ? np : nq ? nq : q;
}
int up[M], c[256] = {2}, aa[M];
vector<int> G[M];
void rsort() {
    FOR (i, 1, 256) c[i] = 0;
    FOR (i, 2, sz) up[i] = *(one[i] + len[fa[i]]);
    FOR (i, 2, sz) c[up[i]]++;
    FOR (i, 1, 256) c[i] += c[i - 1];
    FOR (i, 2, sz) aa[--c[up[i]]] = i;
    FOR (i, 2, sz) G[fa[aa[i]]].push_back(aa[i]);
}
```

- 匹配

```
int u = 1, l = 0;
FOR (i, 0, strlen(s)) {
    int ch = s[i] - 'a';
    while (u && !t[u][ch]) { u = fa[u]; l = len[u]; }
    ++l; u = t[u][ch];
```

```
    if (!u) u = 1;
    if (l) // do something...
}
```

- 获取子串状态

```
int get_state(int l, int r) {
    int u = rpos[r], s = r - l + 1;
    FORD (i, SP - 1, -1) if (len[pa[u][i]] >= s) u = pa[u][i];
    return u;
}
```

- 配合 LCT

```
namespace lct_sam {
    extern struct P *const null;
    const int M = N;
    struct P {
        P *fa, *ls, *rs;
        int last;

        bool has_fa() { return fa->ls == this || fa->rs == this; }
        bool d() { return fa->ls == this; }
        P*& c(bool x) { return x ? ls : rs; }
        P* up() { return this; }
        void down() {
            if (ls != null) ls->last = last;
            if (rs != null) rs->last = last;
        }
        void all_down() { if (has_fa()) fa->all_down(); down(); }
    } *const null = new P{0, 0, 0, 0}, pool[M], *pit = pool;
    P* G[N];
    int t[M][26], len[M] = {-1}, fa[M], sz = 2, last = 1;

    void rot(P* o) {
        bool dd = o->d();
        P *f = o->fa, *t = o->c(!dd);
        if (f->has_fa()) f->fa->c(f->d()) = o; o->fa = f->fa;
        if (t != null) t->fa = f; f->c(dd) = t;
        o->c(!dd) = f->up(); f->fa = o;
    }
    void splay(P* o) {
        o->all_down();
        while (o->has_fa()) {
            if (o->fa->has_fa())
                rot(o->d() ^ o->fa->d() ? o : o->fa);
            rot(o);
        }
        o->up();
    }
    void access(int last, P* u, P* v = null) {
        if (u == null) { v->last = last; return; }
        splay(u);
        P *t = u;
        while (t->ls != null) t = t->ls;
        int L = len[fa[t - pool]] + 1, R = len[u - pool];

        if (u->last) bit::add(u->last - R + 2, u->last - L + 2, 1);
        else bit::add(1, 1, R - L + 1);
        bit::add(last - R + 2, last - L + 2, -1);

        u->rs = v;
        access(last, u->up()->fa, u);
    }
    void insert(P* u, P* v, P* t) {
        if (v != null) { splay(v); v->rs = null; }
        splay(u);
        u->fa = t; t->fa = v;
    }

    void ins(int ch, int pp) {
        int p = last, np = last = sz++;
```

```
        len[np] = len[p] + 1;
        for (; p && !t[p][ch]; p = fa[p]) t[p][ch] = np;
        if (!p) fa[np] = 1;
        else {
            int q = t[p][ch];
            if (len[p] + 1 == len[q]) { fa[np] = q; G[np]->fa = G[q]; }
            else {
                int nq = sz++; len[nq] = len[p] + 1;
                memcpy(t[nq], t[q], sizeof t[0]);
                insert(G[q], G[fa[q]], G[nq]);
                G[nq]->last = G[q]->last;
                fa[nq] = fa[q];
                fa[np] = fa[q] = nq;
                G[np]->fa = G[nq];
                for (; t[p][ch] == q; p = fa[p]) t[p][ch] = nq;
            }
        }
        access(pp + 1, G[np]);
    }

    void init() {
        ++pit;
        FOR (i, 1, N) {
            G[i] = pit++;
            G[i]->ls = G[i]->rs = G[i]->fa = null;
        }
        G[1] = null;
    }
}
```

### 回文自动机

- len 是回文串长度
- num 是该结点表示的前缀的回文后缀个数

```
int nod, lst, rn;
char rs[N];
struct Node { int c[26], len, fa, num; } t[N];
void init() { // 注意 init 不会清空数组，多测需要自行清空。
    t[0].len = 0, t[1].len = -1, t[0].fa = 1;
    rs[0] = -1;
    nod = 1, lst = 0;
}
int getfa(int o) {
    while (rs[rn - t[o].len - 1] != rs[rn]) o = t[o].fa;
    return o;
}
void extend(int x) {
    rs[++rn] = x;
    int p = getfa(lst);
    if (!t[p].c[x]) {
        int np = ++nod;
        t[np].len = t[p].len + 2;
        t[np].fa = t[getfa(t[p].fa)].c[x];
        t[np].num = t[t[np].fa].num + 1;
        t[p].c[x] = np;
    }
    lst = t[p].c[x];
}
```

### manacher

```
// 注意 N 应该是字符串长度的二倍
// 注意字符串从 1 开始编号
int pal[N];
void init(char *s) {
    for (int i = n; i; --i) s[i * 2] = s[i], s[i * 2 + 1] = '#';
    s[0] = s[1] = '#';
    n = 2 * n + 1;
}
```

```
void manacher(char *s) {
    int p = 0, pos = 0; // p 是当前右端点最大的回文串，pos 是这个回文串的中心。
    for (int i = 1; i <= n; ++i) {
        if (p > i) pal[i] = min(p - i + 1, pal[(pos << 1) - i]);
        else pal[i] = 1;
        while (s[i + pal[i]] == s[i - pal[i]]) ++pal[i];
        if (smax(p, i + pal[i] - 1)) pos = i;
    }
}
```

## 哈希

内置了自动双哈希开关（小心 TLE）。

```
#include <bits/stdc++.h>
using namespace std;

#define ENABLE_DOUBLE_HASH

typedef long long LL;
typedef unsigned long long ULL;

const int x = 135;
const int N = 4e5 + 10;
const int p1 = 1e9 + 7, p2 = 1e9 + 9;
ULL xp1[N], xp2[N], xp[N];

void init_xp() {
    xp1[0] = xp2[0] = xp[0] = 1;
    for (int i = 1; i < N; ++i) {
        xp1[i] = xp1[i - 1] * x % p1;
        xp2[i] = xp2[i - 1] * x % p2;
        xp[i] = xp[i - 1] * x;
    }
}

struct String {
    char s[N];
    int length, subsize;
    bool sorted;
    ULL h[N], hl[N];

    ULL hash() {
        length = strlen(s);
        ULL res1 = 0, res2 = 0;
        h[length] = 0;  // ATTENTION!
        for (int j = length - 1; j >= 0; --j) {
        #ifdef ENABLE_DOUBLE_HASH
            res1 = (res1 * x + s[j]) % p1;
            res2 = (res2 * x + s[j]) % p2;
            h[j] = (res1 << 32) | res2;
        #else
            res1 = res1 * x + s[j];
            h[j] = res1;
        #endif
            // printf("%llu\n", h[j]);
        }
        return h[0];
    }

    // 获取子串哈希，左闭右开区间
    ULL get_substring_hash(int left, int right) const {
        int len = right - left;
    #ifdef ENABLE_DOUBLE_HASH
        // get hash of s[left...right-1]
        unsigned int mask32 = ~(0u);
        ULL left1 = h[left] >> 32, right1 = h[right] >> 32;
        ULL left2 = h[left] & mask32, right2 = h[right] & mask32;
        return (((left1 - right1 * xp1[len] % p1 + p1) % p1) << 32) |
               (((left2 - right2 * xp2[len] % p2 + p2) % p2));
    #else
```

```cpp
        return h[left] - h[right] * xp[len];
    #endif
    }

    void get_all_subs_hash(int sublen) {
        subsize = length - sublen + 1;
        for (int i = 0; i < subsize; ++i)
            hl[i] = get_substring_hash(i, i + sublen);
        sorted = 0;
    }

    void sort_substring_hash() {
        sort(hl, hl + subsize);
        sorted = 1;
    }

    bool match(ULL key) const {
        if (!sorted) assert (0);
        if (!subsize) return false;
        return binary_search(hl, hl + subsize, key);
    }

    void init(const char *t) {
        length = strlen(t);
        strcpy(s, t);
    }
};

int LCP(const String &a, const String &b, int ai, int bi) {
    // Find LCP of a[ai...] and b[bi...]
    int l = 0, r = min(a.length - ai, b.length - bi);
    while (l < r) {
        int mid = (l + r + 1) / 2;
        if (a.get_substring_hash(ai, ai + mid) == b.get_substring_hash(bi, bi + mid))
            l = mid;
        else r = mid - 1;
    }
    return l;
}

int check(int ans) {
    if (T.length < ans) return 1;
    T.get_all_subs_hash(ans); T.sort_substring_hash();
    for (int i = 0; i < S.length - ans + 1; ++i)
        if (!T.match(S.get_substring_hash(i, i + ans)))
            return 1;
    return 0;
}

int main() {
    init_xp();  // DON'T FORGET TO DO THIS!

    for (int tt = 1; tt <= kases; ++tt) {
        scanf("%d", &n); scanf("%s", str);
        S.init(str);
        S.hash(); T.hash();
    }
}
```

二维哈希

```cpp
struct Hash2D { // 1-index
    static const LL px = 131, py = 233, MOD = 998244353;
    static LL pwx[N], pwy[N];
    int a[N][N];
    LL hv[N][N];
    static void init_xp() {
        pwx[0] = pwy[0] = 1;
        FOR (i, 1, N) {
            pwx[i] = pwx[i - 1] * px % MOD;
            pwy[i] = pwy[i - 1] * py % MOD;
        }
```

```
    }
    void init_hash(int n, int m) {
        FOR (i, 1, n + 1) {
            LL s = 0;
            FOR (j, 1, m + 1) {
                s = (s * py + a[i][j]) % MOD;
                hv[i][j] = (hv[i - 1][j] * px + s) % MOD;
            }
        }
    }
    LL h(int x, int y, int dx, int dy) {
        --x; --y;
        LL ret = hv[x + dx][y + dy] + hv[x][y] * pwx[dx] % MOD * pwy[dy]
                 - hv[x][y + dy] * pwx[dx] - hv[x + dx][y] * pwy[dy];
        return (ret % MOD + MOD) % MOD;
    }
} ha, hb;
LL Hash2D::pwx[N], Hash2D::pwy[N];
```

### 后缀数组

- hkk 版的后缀数组（写得比较麻烦）

```
// 注意字符串从 1 开始编号
int sa[N], rk[N], _sec[N], fre[N], h[N];
void makeSA() {
    int m = max(n, 127);
    int *rank = rk, *sec = _sec;
    for (int i = 1; i <= m; ++i) fre[i] = 0;
    for (int i = 1; i <= n; ++i) fre[rank[i] = s[i]]++;
    for (int i = 1; i <= m; ++i) fre[i] += fre[i - 1];
    for (int i = n; i >= 1; --i) sa[fre[rank[i]]--] = i;
    for (int k = 1; k <= n; k <<= 1) {
        int p = 0;
        for (int i = n - k + 1; i <= n; ++i) sec[++p] = i;
        for (int i = 1; i <= n; ++i) if (sa[i] > k) sec[++p] = sa[i] - k;

        for (int i = 1; i <= m; ++i) fre[i] = 0;
        for (int i = 1; i <= n; ++i) fre[rank[sec[i]]]++;
        for (int i = 1; i <= m; ++i) fre[i] += fre[i - 1];
        for (int i = n; i >= 1; --i) sa[fre[rank[sec[i]]]--] = sec[i];

        swap(rank, sec);
        p = rank[sa[1]] = 1;
        for (int i = 2; i <= n; ++i)
            rank[sa[i]] = (sec[sa[i]] == sec[sa[i - 1]] && sec[sa[i] + k] == sec[sa[i - 1] + k] ? p : ++p);
        if (p >= n) break;
    }
    for (int i = 1; i <= n; ++i) rk[sa[i]] = i;
}
void getheight() {
    for (int i = 1, k = 0; i <= n; ++i) {
        if (k) --k;
        int j = sa[rk[i] - 1];
        while (s[i + k] == s[j + k]) ++k;
        h[rk[i]] = k;
    }
}
```

- 原模板：构造时间：$O(L \log L)$；查询时间 $O(\log L)$。`suffix` 数组是排好序的后缀下标，`suffix` 的反数组是后缀数组。

```
#include <bits/stdc++.h>
using namespace std;

const int N = 2e5 + 10;
const int Nlog = 18;

struct SuffixArray {
    const int L;
    vector<vector<int> > P;
    vector<pair<pair<int, int>, int> > M;
```

```
    int s[N], sa[N], rank[N], height[N];
    // s: raw string
    // sa[i]=k: s[k...L-1] ranks i (0 based)
    // rank[i]=k: the rank of s[i...L-1] is k (0 based)
    // height[i] = lcp(sa[i-1], sa[i])

    SuffixArray(const string &raw_s) : L(raw_s.length()), P(1, vector<int>(L, 0)), M(L) {
        for (int i = 0; i < L; i++)
            P[0][i] = this->s[i] = int(raw_s[i]);
        for (int skip = 1, level = 1; skip < L; skip *= 2, level++) {
            P.push_back(vector<int>(L, 0));
            for (int i = 0; i < L; i++)
                M[i] = make_pair(make_pair(P[level - 1][i], i + skip < L ? P[level - 1][i + skip] : -1000), i);
            sort(M.begin(), M.end());
            for (int i = 0; i < L; i++)
                P[level][M[i].second] = (i > 0 && M[i].first == M[i - 1].first) ? P[level][M[i - 1].second] : i;
        }
        for (unsigned i = 0; i < P.back().size(); ++i) {
            rank[i] = P.back()[i];
            sa[rank[i]] = i;
        }
    }

    // This is a traditional way to calculate LCP
    void getHeight() {
        memset(height, 0, sizeof height);
        int k = 0;
        for (int i = 0; i < L; ++i) {
            if (rank[i] == 0) continue;
            if (k) k--;
            int j = sa[rank[i] - 1];
            while (i + k < L && j + k < L && s[i + k] == s[j + k]) ++k;
            height[rank[i]] = k;
        }
        rmq_init(height, L);
    }

    int f[N][Nlog];
    inline int highbit(int x) {
        return 31 - __builtin_clz(x);
    }

    int rmq_query(int x, int y) {
        int p = highbit(y - x + 1);
        return min(f[x][p], f[y - (1 << p) + 1][p]);
    }

    // arr has to be 0 based
    void rmq_init(int *arr, int length) {
        for (int x = 0; x <= highbit(length); ++x)
            for (int i = 0; i <= length - (1 << x); ++i) {
                if (!x) f[i][x] = arr[i];
                else f[i][x] = min(f[i][x - 1], f[i + (1 << (x - 1))][x - 1]);
            }
    }

    #ifdef NEW
    // returns the length of the longest common prefix of s[i...L-1] and s[j...L-1]
    int LongestCommonPrefix(int i, int j) {
        int len = 0;
        if (i == j) return L - i;
        for (int k = (int) P.size() - 1; k >= 0 && i < L && j < L; k--) {
            if (P[k][i] == P[k][j]) {
                i += 1 << k;
                j += 1 << k;
                len += 1 << k;
            }
        }
        return len;
    }
    #else
```

```cpp
    int LongestCommonPrefix(int i, int j) {
        // getHeight() must be called first
        if (i == j) return L - i;
        if (i > j) swap(i, j);
        return rmq_query(i + 1, j);
    }
    #endif

    int checkNonOverlappingSubstring(int K) {
        // check if there is two non-overlapping identical substring of length K
        int minsa = 0, maxsa = 0;
        for (int i = 0; i < L; ++i) {
            if (height[i] < K) {
                minsa = sa[i]; maxsa = sa[i];
            } else {
                minsa = min(minsa, sa[i]);
                maxsa = max(maxsa, sa[i]);
                if (maxsa - minsa >= K) return 1;
            }
        }
        return 0;
    }

    int checkBelongToDifferentSubstring(int K, int split) {
        int minsa = 0, maxsa = 0;
        for (int i = 0; i < L; ++i) {
            if (height[i] < K) {
                minsa = sa[i]; maxsa = sa[i];
            } else {
                minsa = min(minsa, sa[i]);
                maxsa = max(maxsa, sa[i]);
                if (maxsa > split && minsa < split) return 1;
            }
        }
        return 0;
    }

} *S;

int main() {
    string s, t;
    cin >> s >> t;
    int sp = s.length();
    s += "*" + t;
    S = new SuffixArray(s);
    S->getHeight();
    int left = 0, right = sp;
    while (left < right) {
        int mid = (left + right + 1) / 2;
        if (S->checkBelongToDifferentSubstring(mid, sp))
            left = mid;
        else right = mid - 1;
    }
    printf("%d\n", left);
}
```

- SA-IS
- 仅在后缀自动机被卡内存或者卡常且需要 O(1) LCA 的情况下使用（比赛中敲这个我觉得不行）
- UOJ 35

```cpp
// rk [0..n-1] -> [1..n], sa/ht [1..n]
// s[i] > 0 && s[n] = 0
// b: normally as bucket
// c: normally as bucket1
// d: normally as bucket2
// f: normally as cntbuf

template<size_t size>
struct SuffixArray {
    bool t[size << 1];
    int b[size], c[size];
```

```cpp
    int sa[size], rk[size], ht[size];
    inline bool isLMS(const int i, const bool *t) { return i > 0 && t[i] && !t[i - 1]; }
    template<class T>
    inline void inducedSort(T s, int *sa, const int n, const int M, const int bs,
                            bool *t, int *b, int *f, int *p) {
        fill(b, b + M, 0); fill(sa, sa + n, -1);
        FOR (i, 0, n) b[s[i]]++;
        f[0] = b[0];
        FOR (i, 1, M) f[i] = f[i - 1] + b[i];
        FORD (i, bs - 1, -1) sa[--f[s[p[i]]]] = p[i];
        FOR (i, 1, M) f[i] = f[i - 1] + b[i - 1];
        FOR (i, 0, n) if (sa[i] > 0 && !t[sa[i] - 1]) sa[f[s[sa[i] - 1]]++] = sa[i] - 1;
        f[0] = b[0];
        FOR (i, 1, M) f[i] = f[i - 1] + b[i];
        FORD (i, n - 1, -1) if (sa[i] > 0 && t[sa[i] - 1]) sa[--f[s[sa[i] - 1]]] = sa[i] - 1;
    }
    template<class T>
    inline void sais(T s, int *sa, int n, bool *t, int *b, int *c, int M) {
        int i, j, bs = 0, cnt = 0, p = -1, x, *r = b + M;
        t[n - 1] = 1;
        FORD (i, n - 2, -1) t[i] = s[i] < s[i + 1] || (s[i] == s[i + 1] && t[i + 1]);
        FOR (i, 1, n) if (t[i] && !t[i - 1]) c[bs++] = i;
        inducedSort(s, sa, n, M, bs, t, b, r, c);
        for (i = bs = 0; i < n; i++) if (isLMS(sa[i], t)) sa[bs++] = sa[i];
        FOR (i, bs, n) sa[i] = -1;
        FOR (i, 0, bs) {
            x = sa[i];
            for (j = 0; j < n; j++) {
                if (p == -1 || s[x + j] != s[p + j] || t[x + j] != t[p + j]) { cnt++, p = x; break; }
                else if (j > 0 && (isLMS(x + j, t) || isLMS(p + j, t))) break;
            }
            x = (~x & 1 ? x >> 1 : x - 1 >> 1), sa[bs + x] = cnt - 1;
        }
        for (i = j = n - 1; i >= bs; i--) if (sa[i] >= 0) sa[j--] = sa[i];
        int *s1 = sa + n - bs, *d = c + bs;
        if (cnt < bs) sais(s1, sa, bs, t + n, b, c + bs, cnt);
        else FOR (i, 0, bs) sa[s1[i]] = i;
        FOR (i, 0, bs) d[i] = c[sa[i]];
        inducedSort(s, sa, n, M, bs, t, b, r, d);
    }
    template<typename T>
    inline void getHeight(T s, const int n, const int *sa) {
        for (int i = 0, k = 0; i < n; i++) {
            if (rk[i] == 0) k = 0;
            else {
                if (k > 0) k--;
                int j = sa[rk[i] - 1];
                while (i + k < n && j + k < n && s[i + k] == s[j + k]) k++;
            }
            ht[rk[i]] = k;
        }
    }
    template<class T>
    inline void init(T s, int n, int M) {
        sais(s, sa, ++n, t, b, c, M);
        for (int i = 1; i < n; i++) rk[sa[i]] = i;
        getHeight(s, n, sa);
    }
};

const int N = 2E5 + 100;
SuffixArray<N> sa;

int main() {
    string s; cin >> s; int n = s.length();
    sa.init(s, n, 128);
    FOR (i, 1, n + 1) printf("%d%c", sa.sa[i] + 1, i == _i - 1 ? '\n' : ' ');
    FOR (i, 2, n + 1) printf("%d%c", sa.ht[i], i == _i - 1 ? '\n' : ' ');
}
```

## KMP

- 前缀函数（每一个前缀的最长 border）

```cpp
// 注意字符串从 1 开始编号
int fa[N];
void getfail(char *s) {
    int p = 0;
    fa[1] = 0;
    for (int i = 2; i <= m; ++i) {
        while (p && t[p + 1] != t[i]) p = fa[p];
        if (t[p + 1] == t[i]) ++p;
        fa[i] = p;
    }
}
auto kmp(char *s, char *t) { // s 是文本串, t 是模板串
    int p = 0;
    vector<int> ans;
    for (int i = 1; i <= n; ++i) {
        while (p && t[p + 1] != s[i]) p = fa[p];
        if (t[p + 1] == s[i]) ++p;
        if (p == m) ans.push_back(i - m + 1);
    }
    return ans;
}
```

- Z 函数（每一个后缀和该字符串的 LCP 长度）

```cpp
// 注意字符串从 1 开始编号
int z[N], r[N];
void get_z(char *s, int n) { // z : s 的每一个后缀和 s 的 LCP
    z[1] = 0;
    for (int i = 2, p = 1; i <= n; ++i) {
        if (i + z[i - p + 1] < p + z[p]) z[i] = z[i - p + 1];
        else {
            z[i] = max(0, p + z[p] - i);
            while (i + z[i] <= n && s[i + z[i]] == s[1 + z[i]]) ++z[i];
            p = i;
        }
    }
}
void exkmp(char *s, int n, char *t, int m) { // r : s 的每一个后缀与 t 的 LCP, 此时 z 数组是 t 串的 z 函数
    for (int i = 1, p = 0; i <= n; ++i) {
        if (i + z[i - p + 1] < p + r[p]) r[i]  = z[i - p + 1];
        else {
            r[i] = max(0, p + r[p] - i);
            while (i + r[i] <= n && r[i] + 1 <= m && s[i + r[i]] == t[1 + r[i]]) ++r[i];
            p = i;
        }
    }
}
```

## Trie

```cpp
namespace trie {
    int t[N][26], sz, ed[N];
    void init() { sz = 2; memset(ed, 0, sizeof ed); }
    int _new() { memset(t[sz], 0, sizeof t[sz]); return sz++; }
    void ins(char* s, int p) {
        int u = 1;
        FOR (i, 0, strlen(s)) {
            int c = s[i] - 'a';
            if (!t[u][c]) t[u][c] = _new();
            u = t[u][c];
        }
        ed[u] = p;
    }
}
```

### AC 自动机

```cpp
const int N = 1e6 + 100, M = 26;

int mp(char ch) { return ch - 'a'; }

struct ACA {
    int ch[N][M], danger[N], fail[N];
    int sz;
    void init() {
        sz = 1;
        memset(ch[0], 0, sizeof ch[0]);
        memset(danger, 0, sizeof danger);
    }
    void insert(const string &s, int m) {
        int n = s.size(); int u = 0, c;
        FOR (i, 0, n) {
            c = mp(s[i]);
            if (!ch[u][c]) {
                memset(ch[sz], 0, sizeof ch[sz]);
                danger[sz] = 0; ch[u][c] = sz++;
            }
            u = ch[u][c];
        }
        danger[u] |= 1 << m;
    }
    void build() {
        queue<int> Q;
        fail[0] = 0;
        for (int c = 0, u; c < M; c++) {
            u = ch[0][c];
            if (u) { Q.push(u); fail[u] = 0; }
        }
        while (!Q.empty()) {
            int r = Q.front(); Q.pop();
            danger[r] |= danger[fail[r]];
            for (int c = 0, u; c < M; c++) {
                u = ch[r][c];
                if (!u) {
                    ch[r][c] = ch[fail[r]][c];
                    continue;
                }
                fail[u] = ch[fail[r]][c];
                Q.push(u);
            }
        }
    }
} ac;

char s[N];

int main() {
    int n; scanf("%d", &n);
    ac.init();
    while (n--) {
        scanf("%s", s);
        ac.insert(s, 0);
    }
    ac.build();

    scanf("%s", s);
    int u = 0; n = strlen(s);
    FOR (i, 0, n) {
        u = ac.ch[u][mp(s[i])];
        if (ac.danger[u]) {
            puts("YES");
            return 0;
        }
    }
    puts("NO");
    return 0;
```

```
}
```

# 杂项

## STL

- copy

```
template <class InputIterator, class OutputIterator>
  OutputIterator copy (InputIterator first, InputIterator last, OutputIterator result);
```

- merge (如果相等，第一个优先)

```
template <class InputIterator1, class InputIterator2,
          class OutputIterator, class Compare>
  OutputIterator merge (InputIterator1 first1, InputIterator1 last1,
                        InputIterator2 first2, InputIterator2 last2,
                        OutputIterator result, Compare comp);
```

- for_each

```
template <class InputIterator, class Function>
   Function for_each (InputIterator first, InputIterator last, Function fn);
```

- transform

```
template <class InputIterator, class OutputIterator, class UnaryOperation>
  OutputIterator transform (InputIterator first1, InputIterator last1,
                            OutputIterator result, UnaryOperation op);
```

- numeric_limits

```
template <class T> numeric_limits;
```

- iota

```
template< class ForwardIterator, class T >
void iota( ForwardIterator first, ForwardIterator last, T value );
```

## 日期

```
// Routines for performing computations on dates.  In these routines,
// months are exprsesed as integers from 1 to 12, days are expressed
// as integers from 1 to 31, and years are expressed as 4-digit
// integers.

string dayOfWeek[] = {"Mo", "Tu", "We", "Th", "Fr", "Sa", "Su"};

// converts Gregorian date to integer (Julian day number)

int DateToInt (int m, int d, int y){
  return
    1461 * (y + 4800 + (m - 14) / 12) / 4 +
    367 * (m - 2 - (m - 14) / 12 * 12) / 12 -
    3 * ((y + 4900 + (m - 14) / 12) / 100) / 4 +
    d - 32075;
}

// converts integer (Julian day number) to Gregorian date: month/day/year

void IntToDate (int jd, int &m, int &d, int &y){
  int x, n, i, j;

  x = jd + 68569;
  n = 4 * x / 146097;
  x -= (146097 * n + 3) / 4;
  i = (4000 * (x + 1)) / 1461001;
  x -= 1461 * i / 4 - 31;
  j = 80 * x / 2447;
  d = x - 2447 * j / 80;
  x = j / 11;
```

```
  m = j + 2 - 12 * x;
  y = 100 * (n - 49) + i + x;
}

// converts integer (Julian day number) to day of week

string IntToDay (int jd){
  return dayOfWeek[jd % 7];
}
```

## 子集枚举

- 枚举真子集

```
for (int s = (S - 1) & S; s; s = (s - 1) & S)
```

- 枚举大小为 k 的子集

```
template<typename T>
void subset(int k, int n, T&& f) {
    int t = (1 << k) - 1;
    while (t < 1 << n) {
        f(t);
        int x = t & -t, y = t + x;
        t = ((t & ~y) / x >> 1) | y;
    }
}
```

## 数位 DP

```
LL dfs(LL base, LL pos, LL len, LL s, bool limit) {
    if (pos == -1) return s ? base : 1;
    if (!limit && dp[base][pos][len][s] != -1) return dp[base][pos][len][s];
    LL ret = 0;
    LL ed = limit ? a[pos] : base - 1;
    FOR (i, 0, ed + 1) {
        tmp[pos] = i;
        if (len == pos)
            ret += dfs(base, pos - 1, len - (i == 0), s, limit && i == a[pos]);
        else if (s &&pos < (len + 1) / 2)
            ret += dfs(base, pos - 1, len, tmp[len - pos] == i, limit && i == a[pos]);
        else
            ret += dfs(base, pos - 1, len, s, limit && i == a[pos]);
    }
    if (!limit) dp[base][pos][len][s] = ret;
    return ret;
}

LL solve(LL x, LL base) {
    LL sz = 0;
    while (x) {
        a[sz++] = x % base;
        x /= base;
    }
    return dfs(base, sz - 1, sz - 1, 1, true);
}
```

## 模拟退火

- 最小覆盖圆

```
using LD = double;
const int N = 1E4 + 100;
int x[N], y[N], n;

LD eval(LD xx, LD yy) {
    LD r = 0;
    FOR (i, 0, n)
        r = max(r, sqrt(pow(xx - x[i], 2) + pow(yy - y[i], 2)));
    return r;
```

```
}

mt19937 mt(time(0));
auto rd = bind(uniform_real_distribution<LD>(-1, 1), mt);


int main() {
    int X, Y;
    while (cin >> X >> Y >> n) {
        FOR (i, 0, n) scanf("%d%d", &x[i], &y[i]);
        pair<LD, LD> ans;
        LD M = 1e9;
        FOR (_, 0, 100) {
            LD cur_x = X / 2.0, cur_y = Y / 2.0, T = max(X, Y);
            while (T > 1e-3) {
                LD best_ans = eval(cur_x, cur_y);
                LD best_x = cur_x, best_y = cur_y;
                FOR (___, 0, 20) {
                    LD nxt_x = cur_x + rd() * T, nxt_y = cur_y + rd() * T;
                    LD nxt_ans = eval(nxt_x, nxt_y);
                    if (nxt_ans < best_ans) {
                        best_x = nxt_x; best_y = nxt_y;
                        best_ans = nxt_ans;
                    }
                }
                cur_x = best_x; cur_y = best_y;
                T *= .9;
            }
            if (eval(cur_x, cur_y) < M) {
                ans = {cur_x, cur_y}; M = eval(cur_x, cur_y);
            }
        }
        printf("(%.1f,%.1f).\n%.1f\n", ans.first, ans.second, eval(ans.first, ans.second));
    }
}
```

## 土制 bitset

- 可以用 `auto p = reinterpret_cast<unsigned*>(&x);`（`p[0]` 的最低位就是 `bitset` 的最低位）

```
// M 要开大至少 1 个 64
const int M = (1E4 + 200) / 64;
typedef unsigned long long ULL;
const ULL ONE = 1;

struct Bitset {
    ULL a[M];
    void go(int x) {
        int offset = x / 64; x %= 64;
        for (int i = offset, j = 0; i + 1 < M; ++i, ++j) {
            a[j] |= a[i] >> x;
            if (x) a[j] |= a[i + 1] << (64 - x); // 不能左移 64 位
        }
    }
    void init() { memset(a, 0, sizeof a); }
    void set(int x) {
        int offset = x / 64; x %= 64;
        a[offset] |= (ONE << x);
    }
    void prt() {
        FOR (i, 0, M) FOR (j, 0, 64) putchar((a[i] & (ONE << j)) ? '1' : '0');
        puts("");
    }
    int lowbit() {
        FOR (i, 0, M) if (a[i]) return i * 64 + __builtin_ctzll(a[i]);
        assert (0);
    }
    int highbit(int x) {
        // [0,x) 的最高位
        int offset = x / 64; x %= 64;
        FORD (i, offset, -1) {
```

```
            if (!a[i]) continue;
            if (i == offset) {
                FORD (j, x - 1, -1) if ((ONE << j) & a[i]) { return i * 64 + j; }
            } else return i * 64 + 63 - __builtin_clzll(a[i]);
        }
        assert (0);
    }
};
```

### 随机

- 不要使用 rand()。
- chrono::steady_clock::now().time_since_epoch().count() 可用于计时。
- 64 位可以使用 mt19937_64。

```
int main() {
    mt19937 rng(chrono::steady_clock::now().time_since_epoch().count());
    vector<int> permutation(N);

    for (int i = 0; i < N; i++)
        permutation[i] = i;
    shuffle(permutation.begin(), permutation.end(), rng);

    for (int i = 0; i < N; i++)
        permutation[i] = i;
    for (int i = 1; i < N; i++)
        swap(permutation[i], permutation[uniform_int_distribution<int>(0, i)(rng)]);
```

#### 伪随机数

```
unsigned rnd() {
    static unsigned A = 1 << 16 | 3, B = 33333331, C = 2341;
    return C = A * C + B;
}
```

#### 真实随机数

```
mt19937 mt(time(0));
auto rd = bind(uniform_real_distribution<double>(0, 1), mt);
auto rd2 = bind(uniform_int_distribution<int>(1, 6), mt);
```

#### 随机素数表

42737, 46411, 50101, 52627, 54577, 191677, 194869, 210407, 221831, 241337, 578603, 625409, 713569, 788813, 862481, 2174729, 2326673, 2688877, 2779417, 3133583, 4489747, 6697841, 6791471, 6878533, 7883129, 9124553, 10415371, 11134633, 12214801, 15589333, 17148757, 17997457, 20278487, 27256133, 28678757, 38206199, 41337119, 47422547, 48543479, 52834961, 76993291, 85852231, 95217823, 108755593, 132972461, 171863609, 173629837, 176939899, 207808351, 227218703, 306112619, 311809637, 322711981, 330806107, 345593317, 345887293, 362838523, 373523729, 394207349, 409580177, 437359931, 483577261, 490845269, 512059357, 534387017, 698987533, 764016151, 906097321, 914067307, 954169327

1572869, 3145739, 6291469, 12582917, 25165843, 50331653 （适合哈希的素数）

```
from random import randint

def is_prime(num, test_count):
    if num == 1:
        return False
    if test_count >= num:
        test_count = num - 1
    for x in range(test_count):
        val = randint(1, num - 1)
        if pow(val, num-1, num) != 1:
            return False
    return True

def generate_big_prime(n):
    found_prime = False
    while not found_prime:
```

```
        p = randint(2**(n-1), 2**n)
        if is_prime(p, 1000):
            return p
```

### NTT 素数表

$p = r2^k + 1$，原根是 $g$。

3, 1, 1, 2; 5, 1, 2, 2; 17, 1, 4, 3; 97, 3, 5, 5; 193, 3, 6, 5; 257, 1, 8, 3; 7681, 15, 9, 17; 12289, 3, 12, 11; 40961, 5, 13, 3; 65537, 1, 16, 3; 786433, 3, 18, 10; 5767169, 11, 19, 3; 7340033, 7, 20, 3; 23068673, 11, 21, 3; 104857601, 25, 22, 3; 167772161, 5, 25, 3; 469762049, 7, 26, 3; 1004535809, 479, 21, 3; 2013265921, 15, 27, 31; 2281701377, 17, 27, 3; 3221225473, 3, 30, 5; 75161927681, 35, 31, 3; 77309411329, 9, 33, 7; 206158430209, 3, 36, 22; 2061584302081, 15, 37, 7; 2748779069441, 5, 39, 3; 6597069766657, 3, 41, 5; 39582418599937, 9, 42, 5; 79164837199873, 9, 43, 5; 263882790666241, 15, 44, 7; 1231453023109121, 35, 45, 3; 1337006139375617, 19, 46, 3; 3799912185593857, 27, 47, 5; 4222124650659841, 15, 48, 19; 7881299347898369, 7, 50, 6; 31525197391593473, 7, 52, 3; 180143985094819841, 5, 55, 6; 1945555039024054273, 27, 56, 5; 4179340454199820289, 29, 57, 3.

## Java

### Regex

```java
// Code which demonstrates the use of Java's regular expression libraries.
// This is a solution for
//
//   Loglan: a logical language
//   http://acm.uva.es/p/v1/134.html

import java.util.*;
import java.util.regex.*;

public class LogLan {

    public static void main(String args[]) {

        String regex = BuildRegex();
        Pattern pattern = Pattern.compile(regex);

        Scanner s = new Scanner(System.in);
        while (true) {

            // In this problem, each sentence consists of multiple lines, where the last
            // line is terminated by a period.  The code below reads lines until
            // encountering a line whose final character is a '.'.  Note the use of
            //
            //    s.length() to get length of string
            //    s.charAt() to extract characters from a Java string
            //    s.trim() to remove whitespace from the beginning and end of Java string
            //
            // Other useful String manipulation methods include
            //
            //    s.compareTo(t) < 0 if s < t, lexicographically
            //    s.indexOf("apple") returns index of first occurrence of "apple" in s
            //    s.lastIndexOf("apple") returns index of last occurrence of "apple" in s
            //    s.replace(c,d) replaces occurrences of character c with d
            //    s.startsWith("apple) returns (s.indexOf("apple") == 0)
            //    s.toLowerCase() / s.toUpperCase() returns a new lower/uppercased string
            //
            //    Integer.parseInt(s) converts s to an integer (32-bit)
            //    Long.parseLong(s) converts s to a long (64-bit)
            //    Double.parseDouble(s) converts s to a double

            String sentence = "";
            while (true) {
                sentence = (sentence + " " + s.nextLine()).trim();
                if (sentence.equals("#")) return;
                if (sentence.charAt(sentence.length() - 1) == '.') break;
            }

            // now, we remove the period, and match the regular expression

```

```
            String removed_period = sentence.substring(0, sentence.length() - 1).trim();
            if (pattern.matcher(removed_period).find()) {
                System.out.println("Good");
            } else {
                System.out.println("Bad!");
            }
        }
    }
}
```

### Decimal Format

```
// examples for printing floating point numbers

import java.util.*;
import java.io.*;
import java.text.DecimalFormat;

public class DecFormat {
    public static void main(String[] args) {
        DecimalFormat fmt;

        // round to at most 2 digits, leave of digits if not needed
        fmt = new DecimalFormat("#.##");
        System.out.println(fmt.format(12345.6789)); // produces 12345.68
        System.out.println(fmt.format(12345.0)); // produces 12345
        System.out.println(fmt.format(0.0)); // produces 0
        System.out.println(fmt.format(0.01)); // produces .1

        // round to precisely 2 digits
        fmt = new DecimalFormat("#.00");
        System.out.println(fmt.format(12345.6789)); // produces 12345.68
        System.out.println(fmt.format(12345.0)); // produces 12345.00
        System.out.println(fmt.format(0.0)); // produces .00

        // round to precisely 2 digits, force leading zero
        fmt = new DecimalFormat("0.00");
        System.out.println(fmt.format(12345.6789)); // produces 12345.68
        System.out.println(fmt.format(12345.0)); // produces 12345.00
        System.out.println(fmt.format(0.0)); // produces 0.00

        // round to precisely 2 digits, force leading zeros
        fmt = new DecimalFormat("000000000.00");
        System.out.println(fmt.format(12345.6789)); // produces 000012345.68
        System.out.println(fmt.format(12345.0)); // produces 000012345.00
        System.out.println(fmt.format(0.0)); // produces 000000000.00

        // force leading '+'
        fmt = new DecimalFormat("+0;-0");
        System.out.println(fmt.format(12345.6789)); // produces +12346
        System.out.println(fmt.format(-12345.6789)); // produces -12346
        System.out.println(fmt.format(0)); // produces +0

        // force leading positive/negative, pad to 2
        fmt = new DecimalFormat("positive 00;negative 0");
        System.out.println(fmt.format(1)); // produces "positive 01"
        System.out.println(fmt.format(-1)); // produces "negative 01"

        // qoute special chars (#)
        fmt = new DecimalFormat("text with '#' followed by #");
        System.out.println(fmt.format(12.34)); // produces "text with # followed by 12"

        // always show "."
        fmt = new DecimalFormat("#.#");
        fmt.setDecimalSeparatorAlwaysShown(true);
        System.out.println(fmt.format(12.34)); // produces "12.3"
        System.out.println(fmt.format(12)); // produces "12."
        System.out.println(fmt.format(0.34)); // produces "0.3"

        // different grouping distances:
        fmt = new DecimalFormat("#,####.###");
```

```java
        System.out.println(fmt.format(123456789.123)); // produces "1,2345,6789.123"

        // scientific:
        fmt = new DecimalFormat("0.000E00");
        System.out.println(fmt.format(123456789.123)); // produces "1.235E08"
        System.out.println(fmt.format(-0.000234)); // produces "-2.34E-04"

        // using variable number of digits:
        fmt = new DecimalFormat("0");
        System.out.println(fmt.format(123.123)); // produces "123"
        fmt.setMinimumFractionDigits(8);
        System.out.println(fmt.format(123.123)); // produces "123.12300000"
        fmt.setMaximumFractionDigits(0);
        System.out.println(fmt.format(123.123)); // produces "123"

        // note: to pad with spaces, you need to do it yourself:
        // String out = fmt.format(...)
        // while (out.length() < targlength) out = " "+out;
    }
}
```

### Sort

```java
import java.util.ArrayList;
import java.util.Collections;
import java.util.List;

public class Employee implements Comparable<Employee> {
    private int id;
    private String name;
    private int age;

    public Employee(int id, String name, int age) {
        this.id = id;
        this.name = name;
        this.age = age;
    }

    @Override
    public int compareTo(Employee o) {
        if (id > o.id) {
            return 1;
        } else if (id < o.id) {
            return -1;
        }
        return 0;
    }

    public static void main(String[] args) {
        List<Employee> list = new ArrayList<Employee>();
        list.add(new Employee(2, "Java", 20));
        list.add(new Employee(1, "C", 30));
        list.add(new Employee(3, "C#", 10));
        Collections.sort(list);
    }
}
```

### 扩栈（本地使用）

```c
#include <sys/resource.h>
void init_stack(){
    const rlim_t kStackSize = 512 * 1024 * 1024;
    struct rlimit rl;
    int result;
    result = getrlimit(RLIMIT_STACK, &rl);
    if (result == 0) {
        if (rl.rlim_cur < kStackSize) {
            rl.rlim_cur = kStackSize;
            result = setrlimit(RLIMIT_STACK, &rl);
            if (result != 0) {
```

```
                fprintf(stderr, "setrlimit returned result = %d\n", result);
            }
        }
    }
}
```

### 心态崩了

- `(int)v.size()`
- `1LL << k`
- 递归函数用全局或者 static 变量要小心
- 预处理组合数注意上限
- 想清楚到底是要 `multiset` 还是 `set`
- 提交之前看一下数据范围，测一下边界
- 数据结构注意数组大小（2 倍，4 倍）
- 字符串注意字符集
- 如果函数中使用了默认参数的话，注意调用时的参数个数。
- 注意要读完
- 构造参数无法使用自己
- 树链剖分/dfs 序，初始化或者询问不要忘记 idx, ridx
- 排序时注意结构体的所有属性是不是考虑了
- 不要把 while 写成 if
- 不要把 int 开成 char
- 清零的时候全部用 0~n+1。
- 模意义下不要用除法
- 哈希不要自然溢出
- 最短路不要 SPFA，乖乖写 Dijkstra
- 上取整以及 GCD 小心负数
- mid 用 `l + (r - l) / 2` 可以避免溢出和负数的问题
- 小心模板自带的意料之外的隐式类型转换
- 求最优解时不要忘记更新当前最优解
- 图论问题一定要注意图不连通的问题
- 处理强制在线的时候 lastans 负数也要记得矫正
- 不要觉得编译器什么都能优化
- 分块一定要特判在同一块中的情况